DOBOT

# 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


前書き

1 はじめに

- 1.1 概要
- 1.2 動作環境
- 1.3 ソフトウェアのインストール
- 1.4 本マニュアルの使用方法

2 電源投入及び接続

- 2.1 電源投入
- 2.2 無線接続
- 2.3 有線接続
- 2.4 手動追加

3 インターフェース概要

- 3.1 メイン画面
- 3.2 設定画面
- 3.3 ログ画面
- 3.4 Dobot+画面
- 3.5 適用画面
- 3.6 ジョグ画面
- 3.7 モニタリング画面

4 クイックスタート

5 ロボットの基本操作

- 5.1 ユーザーログイン
- 5.2 イネーブル
- 5.3 モード復元
- 5.4リモートコントロール
  - 5.4.1 デバイスモード
  - 5.4.2 IO/Modbus設定スイッチ
- 5.5 手動/自動モード
- 5.6 ジョグ
- 5.7 ドラッグ
- 5.8 緊急停止及び復帰

5.9 速度調節
5.10 負荷設定
5.11 衝突検出設定
5.12 アラーム対処
6 応用
6.1 応用選択
6.2 ブロックプログラミング
6.2.1 概要
6.2.2 プロジェクト管理
6.2.3 ポイント保存
6.2.4 プログラミング
6.3 スクリプトプログラミング
6.3.1 概要
6.3.2 プロジェクト管理
6.3.3 ポイント保存
6.3.4 プログラミング
6.4 Pythonプログラミング（Magician E6）
6.4.1 概要
6.4.2 プロジェクト管理
6.4.3 ポイント保存
6.4.4 プログラムミング
6.5 デバッグと実行（ブロック/スクリプトプログラミング）
6.6 軌跡復元
7 Dobot+
8 モニタリング
8.1 コントローラーDI/DO
8.1.1 DI/DOモニタリング
8.1.2 I/O設定
8.2 コントローラーAI/AO
8.3 末端I/O
8.4 安全I/O
8.5 Modbus
8.5.1 Modbusモニタリング
8.5.2 Modbus設定

8.6 グローバル変数
8.7 プログラム変数
9 ログ
10 設定
10.1 システム設定
10.2 ユーザー管理
10.3 座標系管理
10.3.1 ユーザー座標系
10.3.2 ツール座標系
10.4 負荷パラメータ
10.5 ボタン設定（Magician E6）
10.6 運動パラメータ
10.6.1 運動パラメータ（CRA）
10.6.2 動作パラメータ（Magician E6）
10.7 姿勢設定
10.8 軌跡再現
10.9 通信設定
10.10 設置設定
10.11 ドラッグ設定
10.12 電源電圧（直流コントローラー/Magician E6）
10.13 安全設定
10.13.1 衝突検出（Magician E6）
10.13.2 安全制限（CRA）
10.13.3 関節リミット（CRA）
10.13.4 安全壁
10.13.5 安全エリア
10.13.6 安全原点
10.13.7 関節ブレーキ
10.14 モード設定
10.15 0点校正
10.16 高度な機能
10.17 ファイル移行
10.18 ファームウェアアップグレード
付録A Modbusレジタス定義

A.1 Modbus説明
A.2 コイルレジスタ（map1、ロボット制御）
A.3 接点レジスタ定義（map1、ロボットステータス）
A.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）
A.5 保持ドレジスタ定義（map1、ロボットとPLC間の通信）
付録B Blocklyプログラミングブロックの説明
B.1 共通説明
ブロックタイプ
運動方式
座標系パラメータ
速度パラメータ
スムーズな遷移
停止条件
B.2 イベントブロックグループ
運転を開始する
サブスレッド起動
B.3 制御ブロックグループ
条件が満たされるまで待機
n回繰り返し
継続的に繰り返し
繰り返しを終了
条件が満たされた場合に実行
条件の成否によってそれぞれ実行
条件が満たされるまで繰り返し実行
ラベルを設定
ラベルにジャンプ
コマンドの折りたたみ
一時停止
停止プログラム
カスタムログ
衝突検知機能を設定
衝突時の後退距離を設定
ユーザー座標系またはツール座標系を変更
ユーザー座標系を計算および更新

ツール座標系を計算および更新
ユーザー座標系を設定
ツール座標系を設定
負荷パラメータを設定
指定時間待機
セーフティウォールのOn/Offを設定
セーフティエリアのOn/Offを設定
システム時間を取得
タイマーを開始
タイマー結果を取得
エンドツールモードを設定
エンドツール485データフォーマットを設定
エンドツール電源のオンオフを設定
カスタムポップアップウィンドウ
コメント
B.4 演算ブロックグループ
四則演算
比較演算
and演算
or演算
not演算
剰余演算
四捨五入演算
単一値演算
プリント
B.5 文字列ブロックグループ
文字列のn番目の文字を取得する
文字列1に文字列2が含まれるか否かを判断する
2つの文字列を結合する
文字列または配列の長さを取得する
2つの文字列を比較する
配列を文字列に変換する
文字列を配列に変換する
指定された配列の要素を取得する

指定された配列の複数の要素を取得する
指定された配列の要素を設定する
B.6 カスタムブロックグループ
グローバル変数を呼び出す
グローバル変数を設定する
カスタム変数を新規作成する
カスタム数値変数
カスタム数値変数の値を設定する
カスタム数値変数の値を増減する
カスタム文字列変数
カスタム文字列変数の値を設定する
配列を作成する
カスタム配列
変換を配列に追加する
指定された配列の項目を削除する
配列のすべての項目を削除する
配列に項目を挿入する
指定された配列の項目を置換する
指定された配列の項目を取得する
配列の項目の総数を取得する
関数を新規作成する
カスタム関数
サブルーチンを新規作成する
サブルーチン
B.7 IOブロックグループ
デジタル出力を設定する
デジタル出力状態を判断する
一連のデジタル出力を設定する
一連のデジタル出力を待つ
デジタル入力を待つ
一連のデジタル入力を待つ
アナログ出力を設定する
アナログ出力を取得する
デジタル入力状態を判断する

アナログ入力を取得する
B.8 モーションブロックグループ
目標点への移動
座標系に沿ったオフセット移動
座標系オフセット後の点を取得
関節オフセット移動
関節オフセット後の点を取得
円弧動作を実行
円周動作を実行
軌跡再現
スムーズな遷移の比率を設定
関節速度の比率を設定
関節加速度の比率を設定
直線速度の比率を設定
直線加速度の比率を設定
グローバル速度の比率を設定
指定点の座標値を変更
指定点の座標値を取得
動作の実現可能性をチェック
指定点の指定座標次元の値を取得
指定点の指定関節の角度を取得
関節角度を正計算で位姿に変換
位姿を逆計算で関節角度に変換
エンコーダの値を取得
B.9 Modbusブロックグループ
Modbusマスターステーションを作成
エンドRS485ベースのModbusマスターステーションを作成
RS485ベースのModbusマスターステーションを作成
マスターステーション作成結果を取得
入力レジスタを待機
保持レジスタを待機
接点レジスタを待機
コイルレジスタを待機
入力レジスタを読み取る

保持レジスタを読み取る
接点レジスタを読み取る
コイルレジスタを読み取る
コイルレジスタを連続して読み取る
保持レジスタを連続して読み取る
コイルレジスタに書き込む
コイルレジスタに連続して書き込む
保持ドレジスタに書き込む
マスターステーションを閉じる
B.10 バスブロックグループ
バスレジスタの値を取得する
バスレジスタの値を設定する
B.11 TCPブロックグループ
SOCKETを接続する
接続したSOCKETの結果を取得する
SOCKETを作成する
作成したSOCKETの結果を取得する
SOCKETを閉じる
変数を受信する
受信した変数の結果を取得する
変数を送信する
送信した変数の結果を取得する
B.12 トレイブロックグループ
トレイを作成する
トレイのポイント総数を取得する
トレイのポイントを取得する
B.13 クイックスタート
Modbusレジスタデータの読み書き
TCP通信によるデータの転送
付録C スクリプトプログラミング関数の説明
C.1 基本文法
基本概念
変数とデータタイプ
演算子

プロセス制御
関数
数学演算によく使われる関数
文字列処理によく使われる関数
テーブル（配列）操作によく使われる関数
C.2 共通説明
運動方式
ポイントパラメータ
座標系パラメータ
速度パラメータ
スムーズな遷移パラメータ
停止条件
IO信号の表現方法
C.3 モーションコマンド
コマンドリスト
MovJ
MovL
Arc
Circle
MovJIO
MovLIO
GetPathStartPose
StartPath
PositiveKin
InverseKin
C.4 相対運動コマンド
コマンドリスト
RelPointUser
RelPointTool
RelMovJTool
RelMovLTool
RelMovJUser
RelMovLUser
RelJointMovJ

RelJoint
C.5 モーションパラメータ
コマンドリスト
CP
VelJ
AccJ
VelL
AccL
SpeedFactor
SetPayload
User
SetUser
CalcUser
Tool
SetTool
CalcTool
GetPose
GetAngle
GetABZ
CheckMovJ
CheckMovL
SetSafeWallEnable
SetWorkZoneEnable
SetCollisionLevel
SetBackDistance
C.6 IO
コマンドリスト
DI
DIGroup
DO
DOGroup
GetDO
GetDOGroup
AI

AO
GetAO
C.7 末端ツール
コマンドリスト
ToolDI
ToolDO
GetToolDO
ToolAI
SetToolMode
GetToolMode
SetToolPower
SetTool485
C.8 TCP&UDP
コマンドリスト
TCPCreate
TCPStart
TCPRead
TCPWrite
TCPDestroy
UDPCreate
UDPRead
UDPWrite
C.9 Modbus
コマンドリスト
ModbusCreate
ModbusRTUCreate
ModbusClose
GetInBits
GetInRegs
GetCoils
SetCoils
GetHoldRegs
SetHoldRegs
C.10 バスレジスタ

コマンドリスト
GetInputBool
GetInputInt
GetInputFloat
GetOutputBool
GetOutputInt
GetOutputFloat
SetOutputBool
SetOutputInt
SetOutputFloat
C.11 プログラム制御
コマンドリスト
Print
Log
Wait
Pause
Halt
ResetElapsedTime
ElapsedTime
Systime
SetGlobalVariable
Popup
C.12 トイレ
コマンドリスト
CreateTray
GetTrayPoint
C.13 セーフスキン
コマンドリスト
EnableSafeSkin
SetSafeSkin
付録D リモコン信号タイムチャート
付録E Pythonプログラミング関数の説明
E.1 基本文法
E.2 共通説明

運動方式
ポイントパラメータ
座標系パラメータ
速度パラメータ
スムーズな遷移パラメータ
IO信号の表現方法
E.3 モーションコマンド
コマンドリスト
MovJ
MovL
Arc
Circle
MovJIO
MovLIO
StartPath
GetPathStartPose
PositiveKin
InverseKin
E.4 相対運動コマンド
コマンドリスト
RelPointUser
RelPointTool
RelMovJTool
RelMovLTool
RelMovJUser
RelMovLUser
RelJointMovJ
RelJoint
E.5 モーションパラメータ
コマンドリスト
CP
VelJ
AccJ
VelL

AccL
SpeedFactor
SetPayload
User
SetUser
CalcUser
Tool
SetTool
CalcTool
GetPose
GetAngle
GetABZ
CheckMovJ
CheckMovL
SetSafeWallEnable
SetWorkZoneEnable
SetCollisionLevel
SetBackDistance
E.6 IO
コマンドリスト
DI
DIGroup
DO
DOGroup
GetDO
GetDOGroup
E.7 末端ツール
コマンドリスト
ToolDI
ToolDO
GetToolDO
SetToolPower
E.8 TCP&UDP
コマンドリスト

TCPCreate
TCPStart
TCPRead
TCPWrite
TCPDestroy
UDPCreate
UDPRead
UDPWrite
E.9 Modbus
コマンドリスト
ModbusCreate
ModbusRTUCreate
ModbusClose
GetInBits
GetInRegs
GetCoils
SetCoils
GetHoldRegs
SetHoldRegs
E.10 プログラム制御
コマンドリスト
Print
Wait
Pause
ResetElapsedTime
ElapsedTime
Systime

# 前書き

## 目的

本マニュアルは、ロボット制御ソフトウェアDobotStudio ProにおけるCRA シリーズおよびMagician E6ロボットの機能や操作方法などを説明し、DobotStudio Proをより簡単に理解し使用できるように目的としています。

## 対象読者

本マニュアルの対象は以下の通り：

- 一般ユーザー
- セールスエンジニア
- 設置及び試運転エンジニア
- テクニカルサポートエンジニア

## 関連ファイル

| ファイル | 説明 | ダウンロード |
| --- | --- | --- |
| Dobot CR Aシリーズハードウェアマニュアル | Dobot CR Aシリーズ協働ロボットの機能、技術仕様、設置ガイドなどを紹介しています。 | Dobotロボット公式サイトにアクセスし、「サービスサポート > ダウンロードセンター」をクリックして、ファイル名で検索するか、製品シリーズで絞り込んでください。 |
| Dobot TCP/IP二次開発インターフェースファイル | TCP/IPプロトコルに基づいた二次開発インターフェースの使用方法を紹介しています。ファイル以外にも、Githubから各言語の二次開発デモを入手できます。 | |
| Dobotバス通信プロトコルマニュアル（EtherNetIP/Profinet） | ロボットのバス通信機能（EtherNetIP/Profinet）の使用方法を紹介しています。 | |
| Dobot協働ロボットティーチペンダントユーザーマニュアル | 協働ロボットに対応するティーチペンダントの使用方法を紹介しています。 | |
| Dobot Magician E6ユーザーマニュアル | Dobot Magician E6協働ロボットの機能、技術仕様、設置ガイドなどを紹介しています。 | |

# 改定履歴

| 日付 | バージョン | 改定内容 |
| --- | --- | --- |
| 2025/01/20 | V4.6.0 | 1.版本デバッグと実行を追加し、デバッグがより便利に<br>2. 軌跡復元を追加し、実行がより柔軟に<br>3. ファイル移行を追加し、設定ファイルのインポート・エクスポートをワンクリックで実現<br>4. ファームウェアのアップグレードを追加し、ファームウェアバージョンをワンクリックでアップグレード<br>5. その他の内容の最適化および画像の更新<br>6. DobotStudio Pro 4.6.0バージョンに更新 |
| 2023/11/28 | V4.5.0 | DobotStudio Pro 4.5.0バージョンに更新 |
| 2023/09/13 | V4.4.1 | 初回リリース、DobotStudio Pro 4.4.1バージョンに対応 |

**説明**
本マニュアルの日本語版は中国語版からの翻訳物です。

# 記号の定義

本マニュアルでは、以下の記号が使用される場合があります。それぞれの意味は以下の通りです。

| 記号 | 説明 |
| --- | --- |
| **危険** | 高度な潜在的危険を示します。これを回避しないと、死亡または重傷を引き起こす可能性があります。 |
| **警告** | 中程度または低程度の潜在的危険を示します。これを回避しないと、軽傷やロボットアームの損傷などを引き起こす可能性があります。 |
| **注意** | 潜在的なリスクを示します。これを無視すると、ロボットアームの損傷、データの損失、または予測不可能な結果を引き起こす可能性があります。 |
| **説明** | 本文の補足情報を示します。本文の内容を強調または補完するものです。 |

# 1 はじめに

## 1.1 概要

DobotStudio Proをご利用いただきありがとうございます。DobotStudio Proは、DobotがDobotロボット向けに開発した制御ソフトウェアです。使いやすさ、実用性の高さ、シンプルで分かりやすいインターフェースが特徴で、ユーザーがDobotロボットの使い方を迅速に習得できるようサポートします。

本文では、DobotStudio Proを使用してCRAシリーズのロボットを制御する方法を主に紹介します。

DobotStudio Proは、PC、Androidタブレット、およびDobot独自開発のティーチペンダントで使用することができます。ティーチペンダント版には、ティーチペンダントのハードウェアと連動する特有の機能（例：イネーブルスイッチ）が搭載されていますが、本文では省略します。詳しくは、『Dobot協働ロボットティーチペンダントユーザーマニュアル』をご参照ください。

## 1.2 動作環境

**DobotStudio Proの動作環境は下記の通り：**

**PC端末**

| スペック | 最小 | 推奨 |
| --- | --- | --- |
| CPU | 64 ビット Intel または AMD、SSE 4.2 または以上、2.9GHz または以上 | |
| OS | • Windows 10（64ビット）Version 1809 または以上<br>• Windows 11 | |
| メモリ (RAM) | 8 GB | 16 GB または以上 |
| グラフィックボード | • DirectX12<br>• 2GBビデオメモリ | • DirectX12<br>• 4GBビデオメモリ、解像度4K または以上 |
| ディスプレイ | 解像度1440 x 900、拡大率100% | 解像度1920 x1080 または以上 |
| ハードディスク | 空き容量4 GB 以上 | • 空き容量4 GB 以上<br>• 内蔵SSD |

**説明:**

- CPU/メモリ/グラフィックボードが最小スペックを下回る場合、フリーズまたはクラッシュする可能性があります。
- ディスプレイ解像度が最低スペックを下回る場合、インターフェースが完全に表示されない可能性があります。
- ソフトウェアをインストールする際に、追加でハードディスクの空き容量が必要になる可能性があります。
- OSが最低スペックを下回る場合、ソフトウェアの交換性の問題が発生する可能性があり、別途で検討する必要があります。

**タブレット端末**

タブレット端末はオプションとしてご購入可能ですが、ご自身で購入する場合、動作環境は下記の通り:

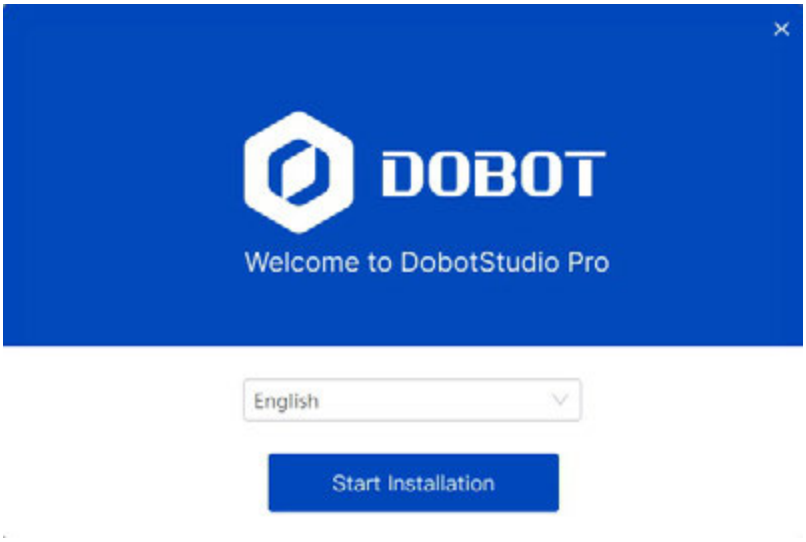
| スペック | 推奨スペック |
| --- | --- |
| CPU | 4 Core |
| OS | Android 10 または以上 |
| メモリ | 2 GB |
| ストレージ | 32 GB |
| モニタ | 8インチ |

# 1.3 ソフトウェアのインストール

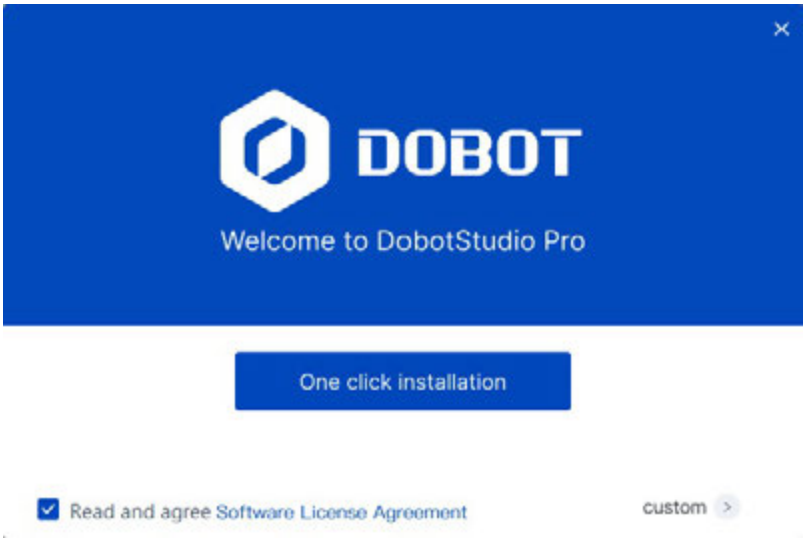
**PC端末**

Dobot公式サイトにより最新のDobotStudio Proのインストールパッケージをダウンロードしインストールしてください。インストール手順は以下の通り:

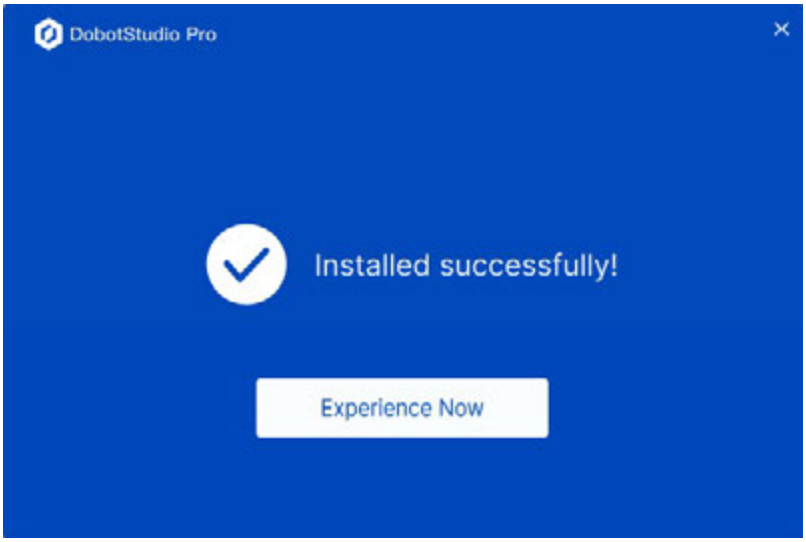
1. インストールパッケージをダブルクリックし、インストール言語を選択して、「**Start Installation**」をクリックします。




2. インストール画面にて「**One click installation**」をクリックし、または「**custom**」をクリックしてインストールパスを設定し、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完了した後、完了画面にて「**Experience Now**」をクリックすると、すぐソフトウェアを開くことができます。




**タブレット端末**

- Android：Dobot公式サイトより最新のDobotStudio Proのインストールパッケージをダウンロードしインストールしてください。

# 1.4 本マニュアルの使用方法

| プロセス | 説明 | 参考章節 |
| --- | --- | --- |
| ロボットに接続 | DobotStudio Proは、有線または無線によるロボットへ接続することができます。ほとんどの機能は、使用前にロボットに接続する必要がありますが、ロボットへ接続していない場合は、システム設定にて表示言語の設定、または端末のクラッシュログをアップロードすることしかできません。 | 電源投入及び接続 |
| ソフトウェアを理解 | DobotStudio Proのメインインターフェイスとその機能をすぐに理解します。 | インターフェース概要 |
| 取付/電圧を設定 | ロボットの取付方法が水平設置ではない場合、先に設置角度を設定する必要があります。ロボットが直流電源入力を使用する場合、電圧範囲を設定する必要があります。 | • 設置設定<br>• 電源電圧 |
| ロボットの機能をすぐに体験 | ロボットが2つポイントの間に往復移動するプログラムを作成して実行します。 | クイックスタート |
| 安全設定 | ロボットを使用する前に、先にリスクアセスメントの結果に基づいてロボッ | • 安全I/O |

| | | |
|---|---|---|
| | トの安全機能を設定してください。 | • 安全設定 |
| ロボットの基本操作を理解 | ユーザーログイン、ジョグ、アラーム処理などのロボットの基本操作を理解します。 | ロボットの基本操作 |
| プログラミング | プログラムを作成することでロボットを制御し、自動で動作させることができます。先に適切なプログラミング方法を選択し、プログラミングインターフェイスの使用方法を学習してください。 | 応用 |
| | 具体的なプログラミングコマンド、コマンドの機能及び使い方を理解します。 | • 付録B Blocklyプログラミングブロックの説明<br>• 付録C スクリプトプログラミング関数の説明<br>• 付録E Pythonプログラミング関数の説明 |
| | IOのリアルタイムのステータスを表示したり、変更したり、IO関連の機能をデバッグしたりすることができます。 | • コントローラーDI/DO<br>• コントローラーAI/AO<br>• 末端I/O |
| エコロジーアクセサリーを使用 | Dobot+プラグインは、Dobotのエコロジーアクセサリーを迅速に設定し使用することに役立ちます。二次開発の手間を省くことができます。Dobotが対応しているエコロジーアクセサリーについて、Dobot公式サイトにご覧ください。 | Dobot+ |
| ソフトウェア及びロボットを設定 | ソフトウェアおよびロボット関連の設定を変更します。ロボット関連の設定はコントローラーに保存されるため、ロボットごとに個別に設定する必要があります。 | 設定 |
| ロボットをリモート制御 | プロジェクトの実行やロボットのステータスの取得など、コントローラーIOを介してロボットをリモート制御することができます。 | コントローラーDI/DO |
| | プロジェクトの実行やロボットのステータスまたはリアルタイムのフィードバックの取得など、Modbus（Modbus-TCPまたはRTU-over-TCP）を介してロボットを制御 | • Modbus<br>• 付録A Modbusレジスタ定義 |

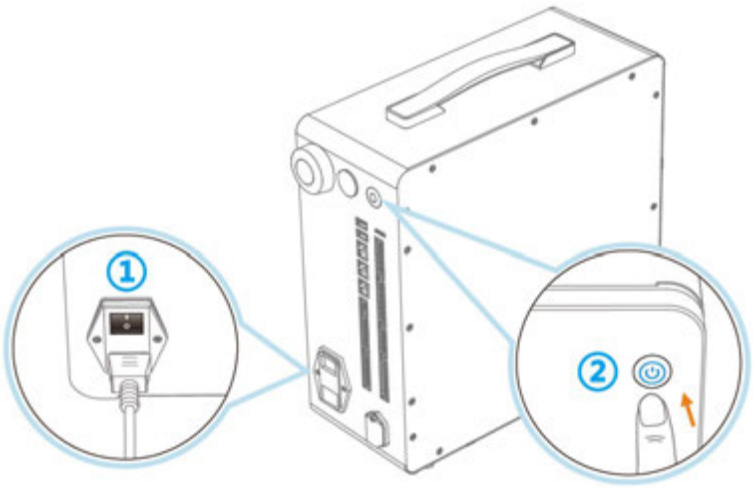
| | | |
|---|---|---|
| | することができます。 | |
| ログの確認とエクスポート | ログはトラブルシューティングに役立ちます。必要な場合、ログをエクスポートして技術サポートに送付し、不具合調査を依頼することができます。 | ログ |
| ファームウェアアップデート | ロボットのファームウェアをワンクリックで最新バージョンにアップデート、またはファームウェアのバージョンをロールバックします。 | ファームウェアアップデート |

# 2 電源投入及び接続

## 2.1 電源投入

### CRAシリーズ

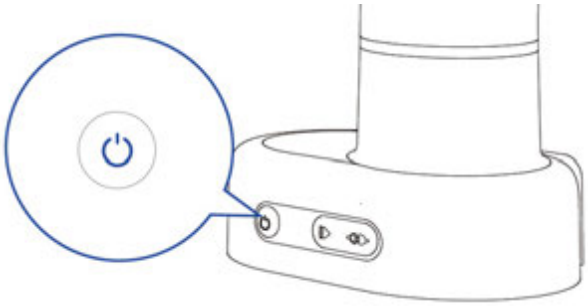
コントローラーを設置して配線を行い、外部電源をオンにした後、電源接続ポートの上にあるスイッチを「|」に切り替え、コントローラーの上にある丸いボタンを短く押します。表示灯が青色に点灯すると、コントローラーが起動完了となります。




**説明:**
CRAシリーズのコントローラーが起動中に、LAN2ポートのランプがオレンジ色に点滅し、コントローラーの中にあるWiFiルーターが初期化していることを示し、LAN、WiFi、ティーチペンダントはロボットに接続することができません。初期化中に、ランプの点滅速度が徐々に速くなり、最終的に消灯になり、初期化完了したことを示し、ロボットに接続することができます。

### Magician E6

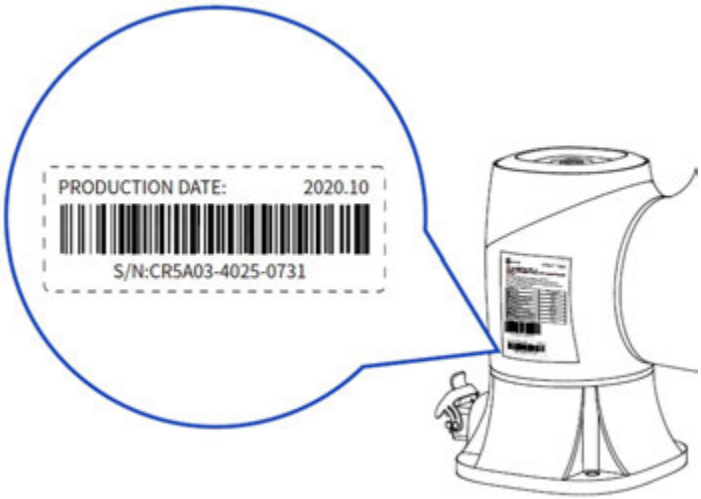
ロボットの設置と配線を完了し、外部電源をオンにした後、ロボットのベースにある丸いボタンを短く押します。ロボットの表示灯が青色の点灯状態になったら、ロボットが起動したことを示します。



## 2.2 無線接続

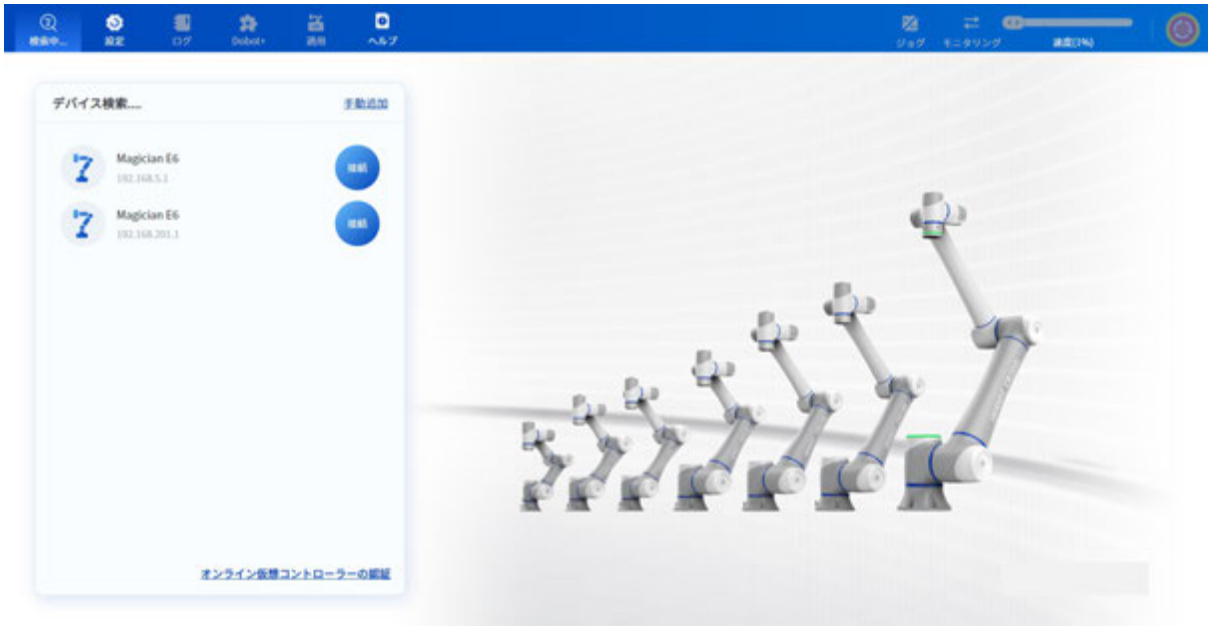
> **説明:**
> Magician E6をワイヤレスで接続する場合、別途でワイヤレスモジュールを購入し、ロボットのUSBポートに挿入してください。

PC端末またはタブレット端末でロボットのWiFiを検索し接続します。WiFiのデフォルトSSIDは「Dobot 製品モデル - シリアル番号」です。シリアル番号は、ロボットの S/N番号 の「-」で接続された 2 つの 4 桁の数字です。（S/N番号はロボットのベースにある銘板に記載してあります。下図のようにCR5Aを例として、デフォルトのSSIDは「DobotCR5A-4025-0731」となります。他もモデルも同様です。）デフォルトのWiFiパスワードは「1234567890」となります。WiFiのSSIDとパスワードは通信設定にて変更可能です。




WiFiで接続する際に、ロボットのIPアドレスは「192.168.201.1」となります。

**PC端末**




DobotStudio Proを開くと、ソフトウェアは自動的に接続可能なロボットを検索し、検索結果が画面に表示されます。コントローラー内部ネットワーク構築の影響で、DobotStudio Proは同じロボットの異なるIPアドレスを複数検出する可能性があります。どのIPアドレスを使用してもロボットに正常に接続することが可能です。「192.168.201.1」に接続することを推奨します。

接続したいロボットの右側にある 

ボタンをクリックし、ロボットに接続します。

**タブレット端末**

画面左上の 

ボタンをクリックし、ロボットに接続します。

> **説明:**
> 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ーのバージョンが一致しているかを確認します:
> - バージョン番号の最初の2桁が一致している場合、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桁が一致していない場合、ソフトウェアはバージョン不一致のポップアップ警告画面が表示され、自動的に接続を切断します。

ロボットに正常に接続した後、ソフトウェアの画面が更新され、接続したロボットの関連情報と3Dモデルが表示されます。ロボットの名称の右側にある切断ボタンをクリックし、現在接続しているロボットの接続を切断することができます。


# 2.3 有線接続

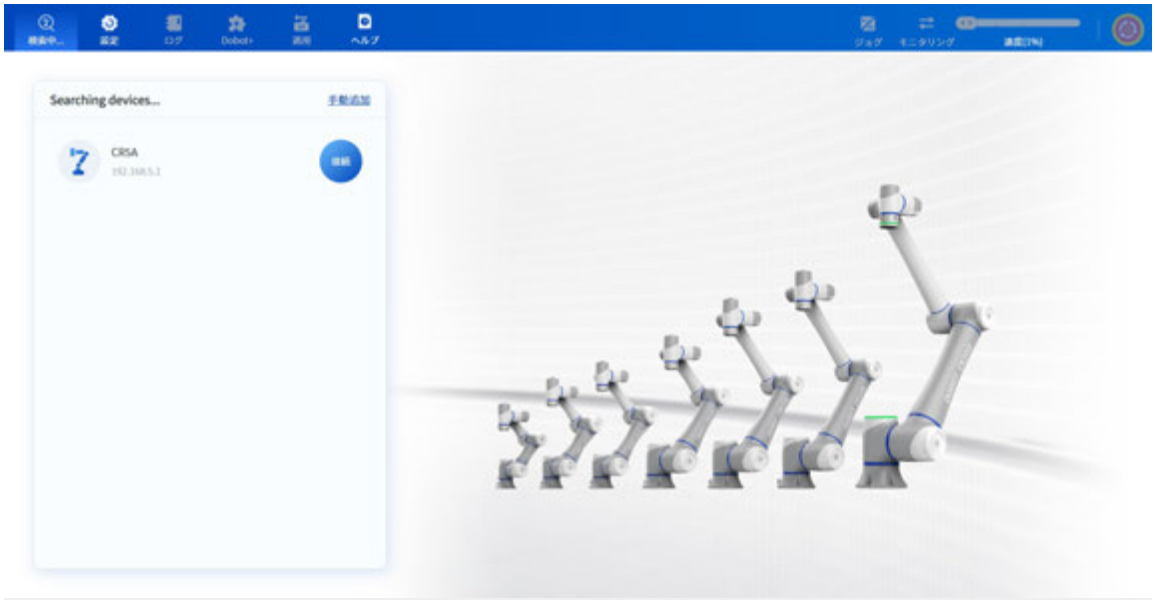
**PC端末のみサポートします。**

LANケーブルの片方をコントローラーのLANポートに接続し、もう片方をPC端末に接続してください。PC端末のIPアドレスを変更し、コントローラーのIPアドレスと同じネットワークセグメントにしてください。LAN1ポートのデフォルトIPアドレスは「192.168.5.1」で、LAN2ポートのデフォルトIPアドレスは「192.168.200.1」です。LAN1ポートのIPアドレスは通信設定にて変更可能ですが、LAN2ポートのIPアドレスは変更不可です。

PC端末のIPアドレス変更方法はWindowsバージョンによって異なります。本文はWindows10を例として紹介します。

1. タスクバーで「ネットワーク」と検索し、「ネットワーク接続の表示」 を選択します。
2. 現在のネットワーク接続（例：イーサネット）のアイコンを右クリックし、「プロパティ」を選択します。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）」** を見つけて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）のプロパティ」 画面で、「次の IP アドレスを使う」 を選択し、PCのIPアドレス、サブネットマスク、および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ーと同じサブネット内の未使用の任意のアドレスに設定し、サブネットマスクとデフォルトゲートウェイをコントローラーと一致させます。例として、PC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。

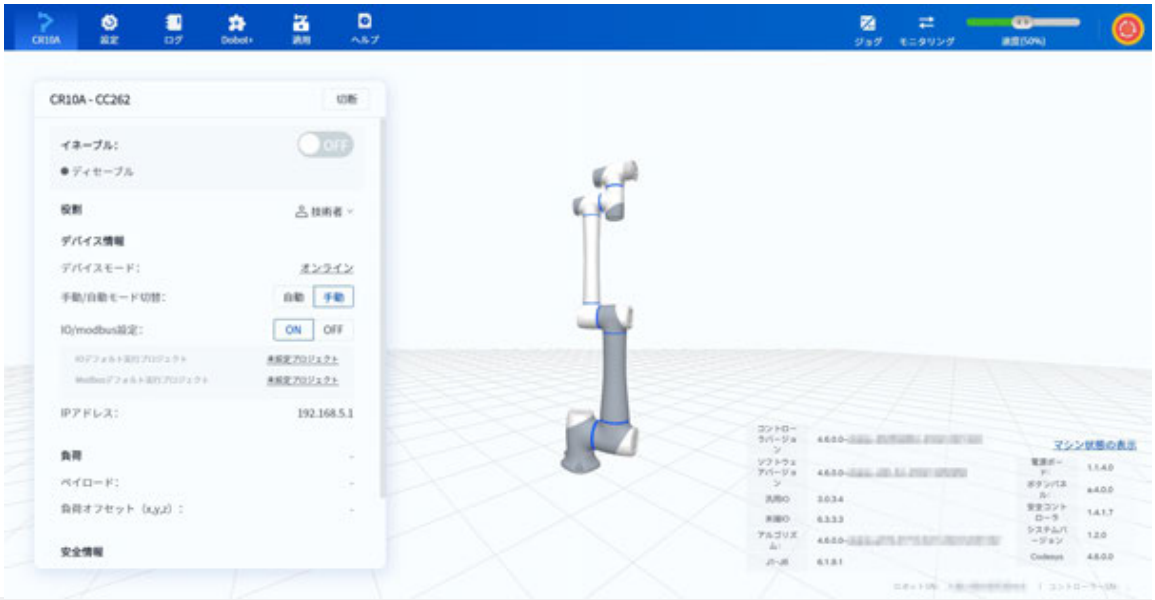

4. 接続したいロボットの右側にある  ボタンをクリックし、ロボットに接続します。




**説明:**
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ーのバージョンが一致しているかを確認します:

- バージョン番号の最初の2桁が一致している場合、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桁が一致していない場合、ソフトウェアはバージョン不一致のポップアップ警告画面が表示され、自動的に接続を切断します。

5. ロボットに正常に接続した後、ソフトウェアの画面が更新され、接続したロボットの関連情報と3Dモデルが表示されます。ロボットの名称の右側にある切断ボタンをクリックし、現在接続しているロボットの接続を切断することができます。

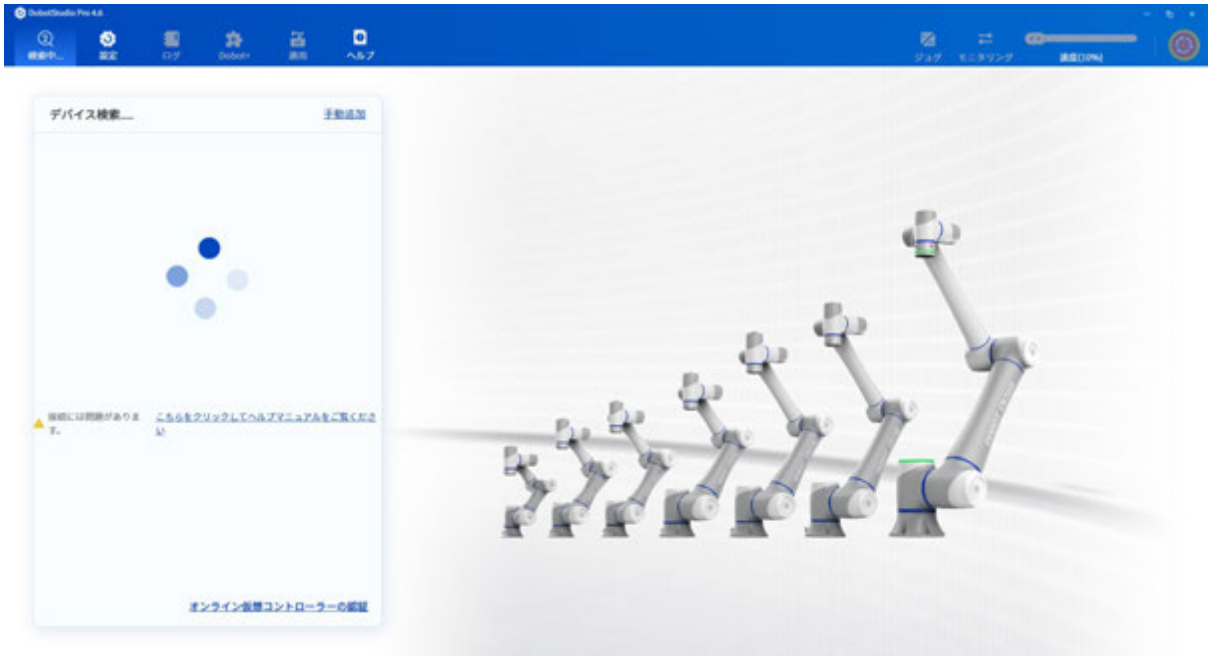



**説明:**
1台のPC端末で複数台のロボットに同時に接続する必要がある場合、ロボットとPC端末を同じローカルエリアネットワークに接続し、PC端末で複数のDobotStudio Proを開き、そ

れぞれのロボットに接続することができます。

## 2.4 手動追加

ロボットが検索されない場合は、手動でロボットのIPアドレスを追加して検索することができます。




DobotStudio Proを起動した後、**「手動追加」** をクリックします。ポップアップページで **「+」** をクリックしてロボットのIPアドレスを追加します。

一度に最大で5つのロボットIPアドレスを手動で追加できます。IPアドレスの入力が完了したら、**「確定」** をクリックすると、手動で追加したロボットが優先的に検索されます。

# 3 インターフェース概要


- 3.1 メイン画面
- 3.2 設定画面
- 3.3 ログ画面
- 3.4 Dobot+画面
- 3.5 適用画面
- 3.6 ジョグ画面
- 3.7 モニタリング画面

# 3.1 メイン画面

ソフトウェアがロボットに接続すると、メイン画面は下記の通り表示されます。


| 番号 | 説明 |
| --- | --- |
| 1 | クリックすると、メイン画面（現在の画面）が表示されます。ロボットのアイコンの色は、ロボットの表示灯と同期し、ロボットの状態に応じて変化します。表示灯の色の定義について、対応するロボットハードウェアマニュアルをご参照ください。 |
| 2 | クリックすると、設定画面が表示され、ロボットの設置設定、運動パラメータ、安全設定などの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画面が表示され、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+画面が表示され、Dobot+プラグインをインストールし使用することができます。 |
| 5 | クリックすると、応用画面が表示され、Blocklyまたはスクリプトによりプログラミングを作成することができます。 |
| 6 | クリックすると、ヘルプドキュメントが表示され、ドキュメントは、ソフトウェアインターフェイスでの表示、独立ウィンドウでの表示、ブラウザでの表示を対応します。 |
| 7 | クリックすると、ジョグ画面が表示され、ロボットのジョグ操作やインチング操作を実行することができます。 |

| | |
|---|---|
| 8 | クリックすると、モニタリング画面が表示され、ロボットのIOやグローバル変数などを監視することができます。 |
| 9 | ロボットの速度を制御するためのグローバル速度比率を設定することができます。 |
| 10 | 緊急停止ボタンです。ロボット動作中に緊急事態が発生した場合、このボタンを押すとロボットが緊急停止します。 |
| 11 | このパネルでは、ロボットイネーブル、ユーザー機能の切り替え、デバイス情報の設定、負荷情報や安全情報（例: セーフティチェックサム情報）を表示します。 |
| 12 | ロボットの3Dモデルで、姿勢は実際のロボットと一致しています。 |
| 13 | ロボットのSNコードとソフト・ファームウェアのバージョン情報です。<br>技術サポートに問題を報告する際は、この部分の情報をスクリーンショットして技術サポートに送信してください。問題の特定が容易になります。<br>ファームウェアバージョンをアップグレードする場合は、優先的にロボットメンテナンスツールを使用してください。または、ファームウェアアップグレードからアップグレードを実行してください。 |
| 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。<br>  |

# セーフティチェックサム

**セーフティチェックサム**は、安全パラメータが特定の検証アルゴリズムによって計算された結果です。安全パラメータが変更された場合、セーフティチェックサムも変更されます。

DobotStudio Proはエラーダイアログを表示します。ユーザーはセーフティチェックサムの不一致問題を解決しない限り、ロボットの使用を続けることはできません。

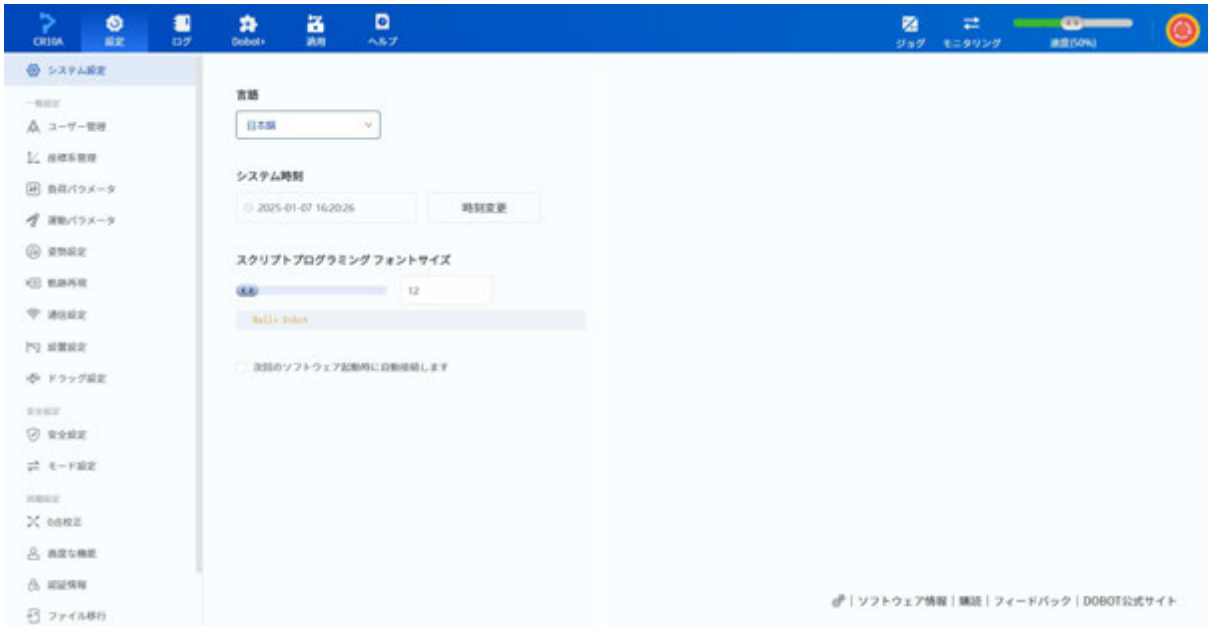

- **現在の安全パラメータの使用**をクリックすると、現在のシステムの安全パラメータに基づいて新しいセーフティチェックサムが生成されます。設定が成功すると、**現在の安全パラメータを正常に設定しました**というメッセージが表示されます。
- **安全パラメータの復元**をクリックすると、前回正しく設定された安全パラメータに復元され、セーフティチェックサムは変更されません。復元が成功すると、**デフォルトの安全パラメータを正常に復元しました**というメッセージが表示されます。

セーフティチェックサムに影響を与える安全パラメータは以下の通りです:

| DobotStudio Pro画面 | パラメータ |
|---|---|
| 安全I/O | 安全I/O画面のすべてのパラメータ |
| 高度な機能 | トルク制限 |
| 安全基準点 | 関節座標 |
| 安全エリア | 安全エリア画面のすべてのパラメータ |
| 安全壁 | 安全壁画面のすべてのパラメータ |
| 衝突検知（Magician E6） | 衝突検知画面のすべてのパラメータ |
| 設置設定 | 傾斜角度、回転角度 |
| 動作パラメータ | 再現速度 |
| モード設定 | 手動モード、自動モード |
| 安全制限 | 安全制限画面のすべてのパラメータ |

# 3.2 設定画面

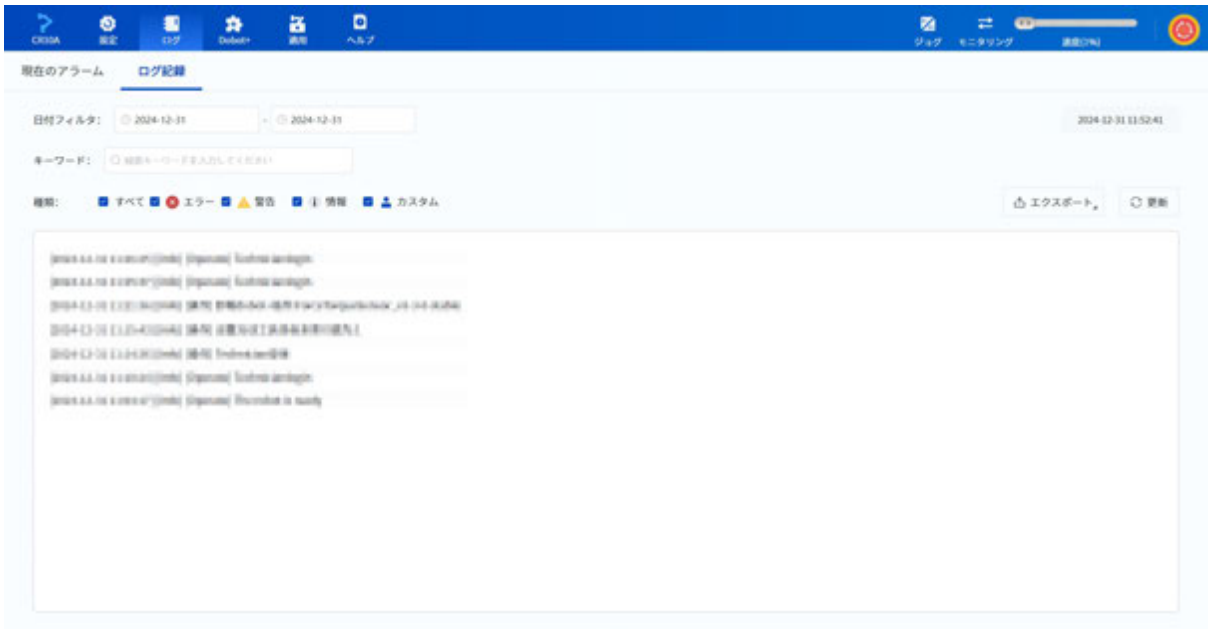
設定画面では、表示言語、ユーザー座標系またはツール座標系、安全設定など、ソフトウェアやロボット関連の設定を変更することができます。詳細は各システム設定の説明をご参照ください。



# 3.3 ログ画面

ログ画面では、**現在のアラーム**と**ログ記録**の二つの画面が含まれます。

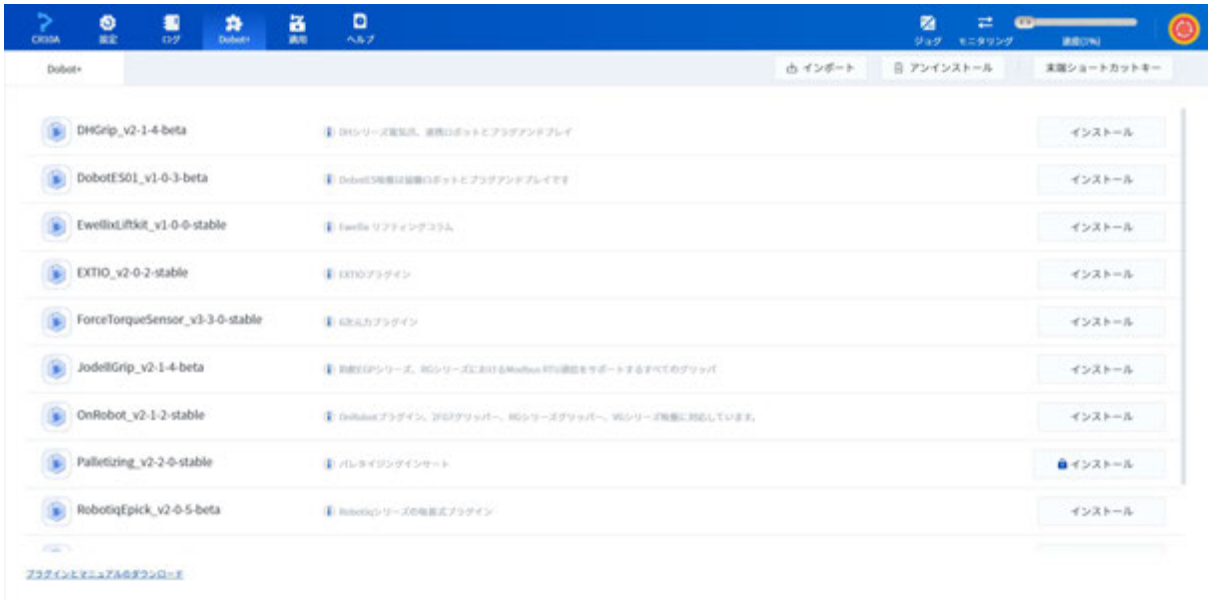
- **現在のアラーム**画面では、アラームの表示及びクリアに使用されます。詳細はアラーム対処をご参照ください。。
- **ログ記録**画面では、ログの表示及びエクスポートに使用されます。詳細はログをご参照ください。



# 3.4 Dobot+画面

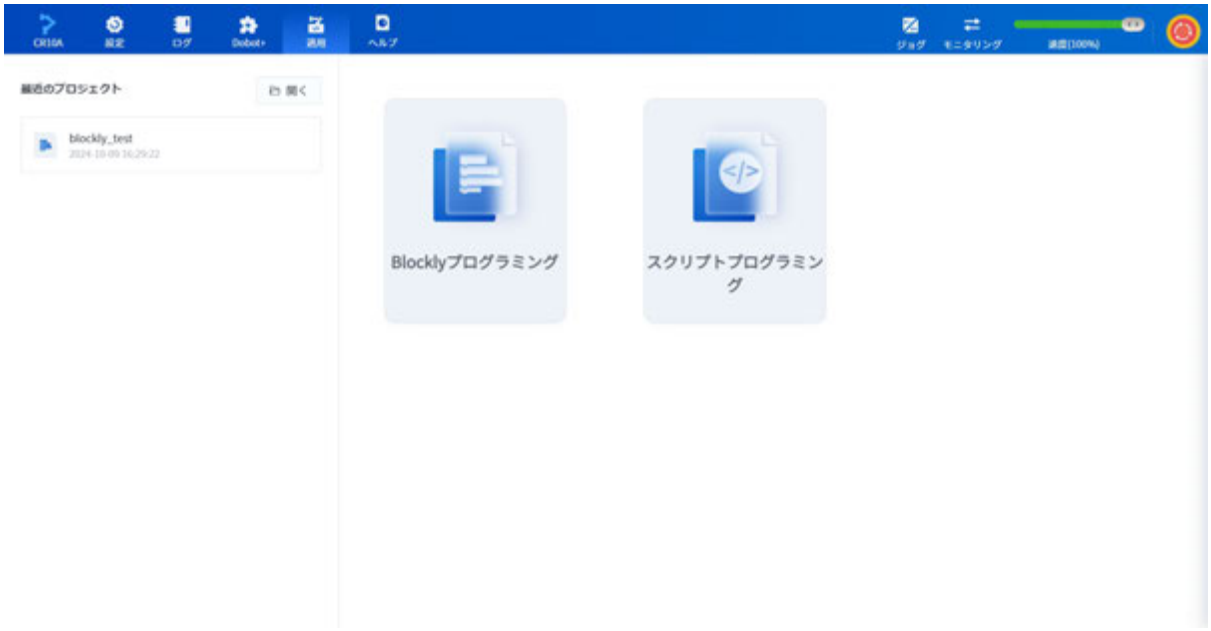
Dobot+画面では、Dobot+プラグインの管理及び使用に使用されます。詳細はDobot+をご参照ください。

Dobot+プラグインは、弊社がエコロジーアクセサリーに対し開発した専用プラグインとなり、二次開発が必要なく、グリッパーなどのエコロジーアクセサリーをプラグイン経由で直接設定して使用することができます。



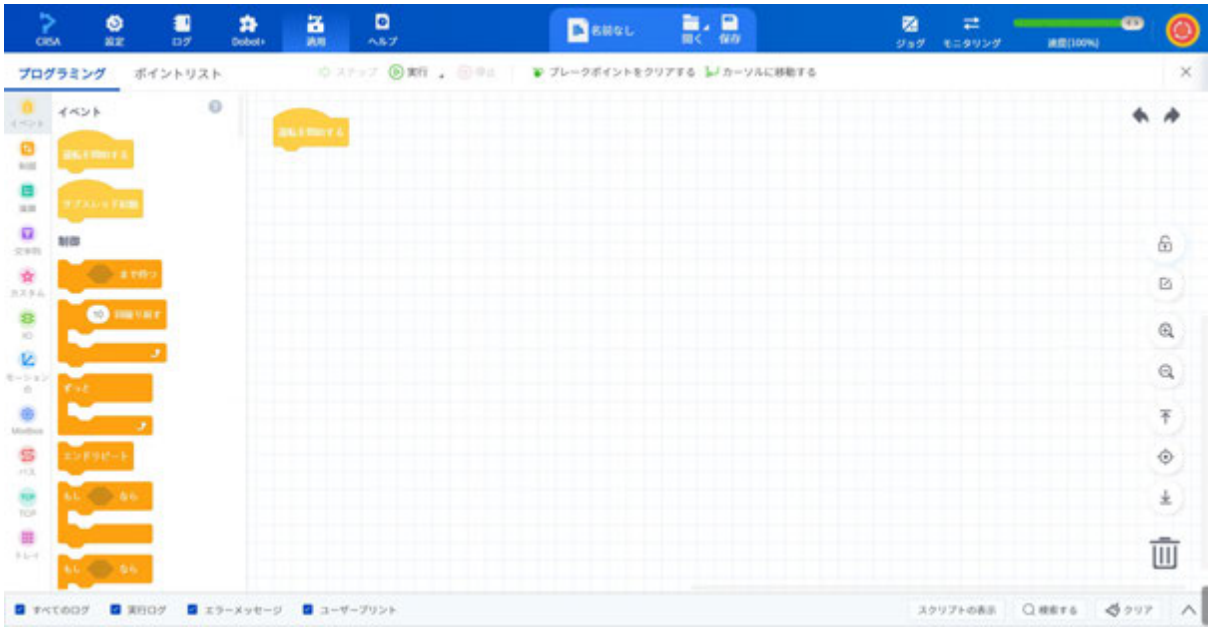
# 3.5 適用画面

適用画面では、プロジェクトの作成や実行を行い、ロボットの自動運転を実現することができます。



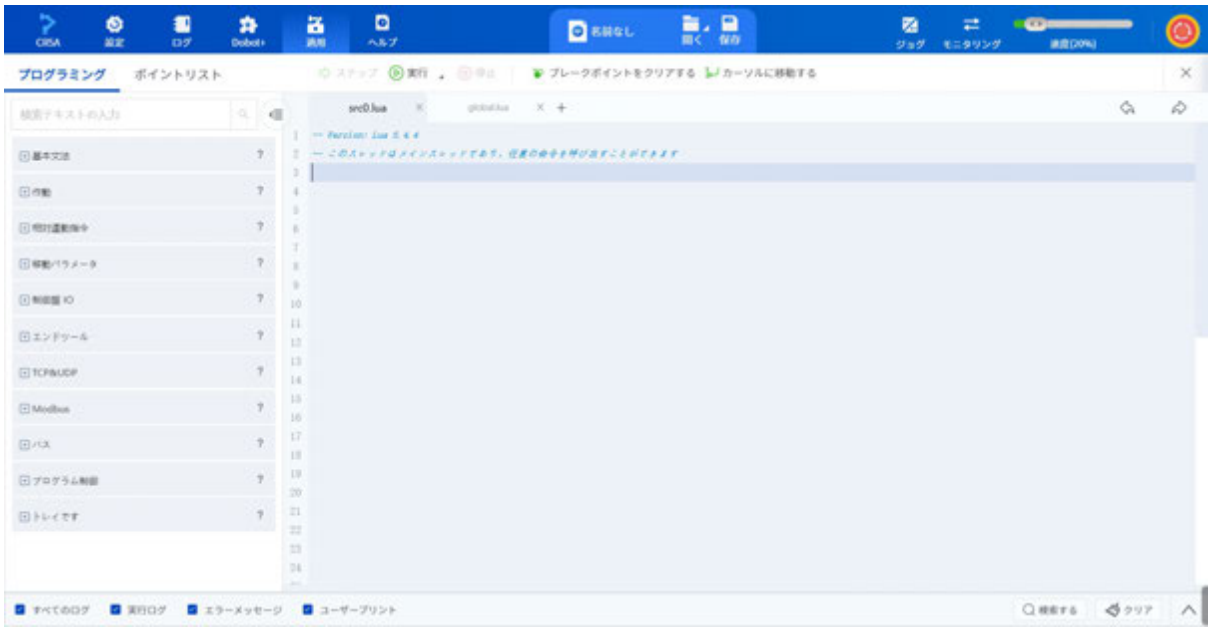

## Blocklyプログラミング

Blocklyプログラミングは、グラフィカルなプログラミング方法です。プログラミング画面の左側にあるブロックを右側のキャンバスにドラッグすることでプログラミングを編集することができます。詳細はBlocklyプログラミングをご参照ください。



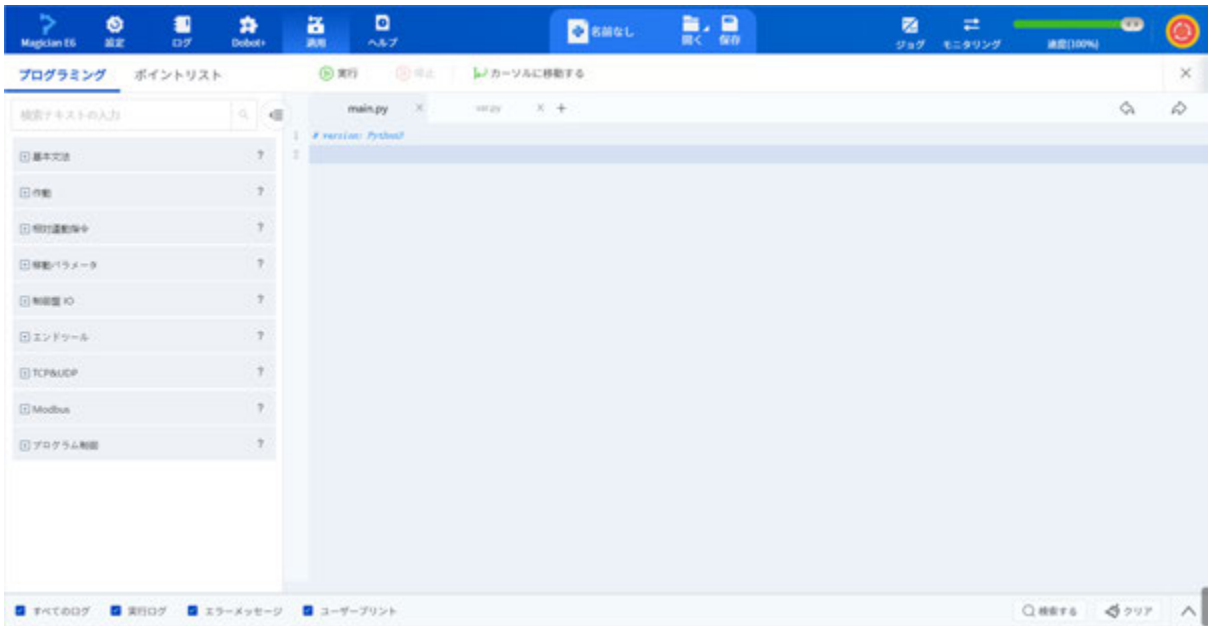
## スクリプトプログラミング

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くプログラミング方法です。プログラミング画面の左側でコマンドを検索し、パラメータを設定した後、右側のプログラミングエリアに追加するか、右側のプログラミングエリアにプログラムを直接に書き込むことができます。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くプログラミング方法です。**PCがMagician E6ロボットに接続されている場合にのみ利用可能**で、教育や研究に使用されます。操作画面は基本的にスクリプトプログラミングと同じですが、デバッグには対応していません。詳細はPythonプログラミングをご参照ください。



# 3.6 ジョグ画面

ジョグ画面では、ロボットのジョグ動作またはインチング動作を実行するために使用されます。関節ジョグ及び座標系ジョグを対応します。詳細はジョグ機能説明をご参照ください。

ジョグ画面の右上にあるをクリックすると、自由にドラッグ可能な独立ウィンドウに変わります。独立ウィンドウを閉じて再度ジョグ画面を開くと、ジョグ画面が埋め込み仕様に戻ります。

# 3.7 モニタリング画面

モニタリング画面では、ロボットの各IOの状態や機能を監視・設定することができます。グローバル変数の管理・表示も可能です。詳細は各モニタリングの使用説明をご参照ください。

モニタリング画面の右上にあるをクリックすると、自由にドラッグ可能な独立ウィンドウに変わります。独立ウィンドウを閉じて再度モニタリング画面を開くと、モニタリング画面が埋め込み仕様に戻ります。

# 4 クイックスタート

本記事では、ロボットが2点間を周期的に移動するBlocklyプログラムの新規作成方法を紹介し、Dobotロボットの機能をすぐ体験できるように役立ちます。

本記事では、目的を実現するための最も簡単な操作手順のみを紹介します。各機能の詳細内容説明について、本マニュアルの他の記事にご参照ください。

## ロボットをイネーブル

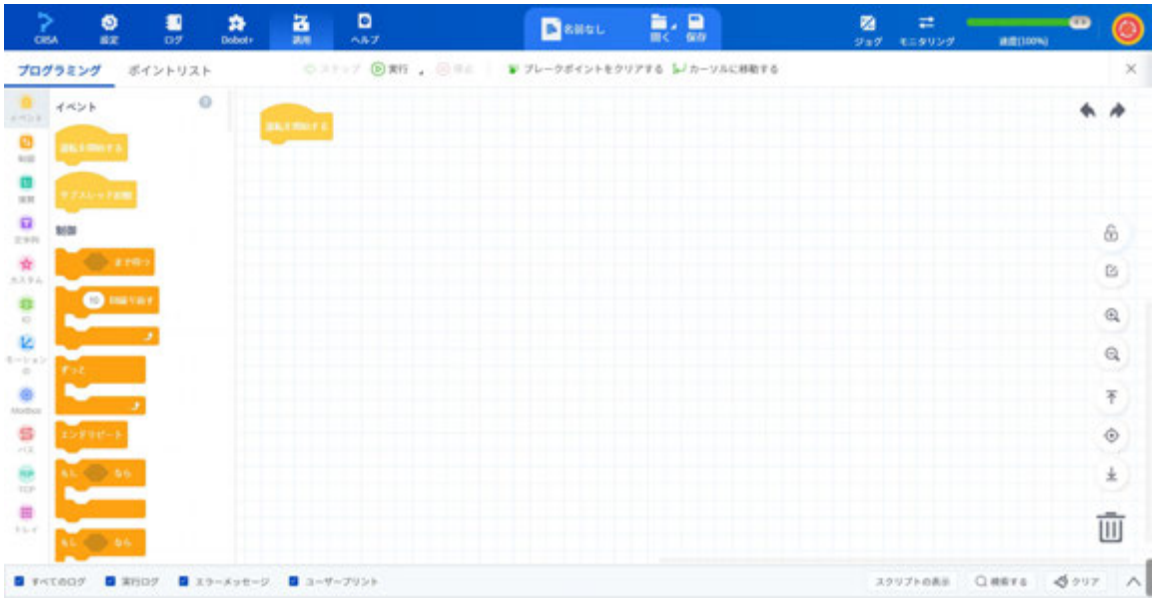
1. ロボットに接続した後、メイン画面の情報パネルにて**イネーブル**スイッチをクリックします。

2. **負荷イネーブル設定**のポップアップ画面が表示されます。ロボットにエンドツールが装着していない場合、そのまま**イネーブルを確定**ボタンをクリックします。ロボットにエンドツールが装着している場合、5.2 イネーブルを参考して負荷パラメータを設定してください。

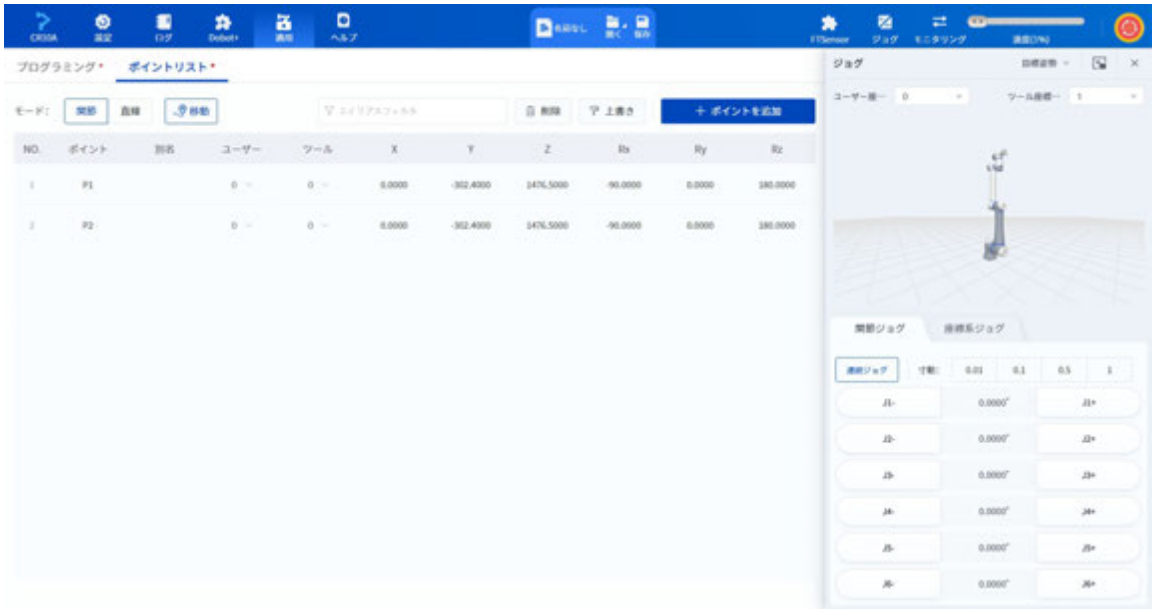

3. イネーブル完了後、イネーブルボタンが**ON**に変わります。


## Blocklyプロジェクトを作成

1. **適応** > **Bloclyプログラミング**画面を開きます。



2. **ポイントリスト**を開きます。



3. **ジョグ**画面の制御ボタン（例えば、**J1+**、**J1-**）を長押ししてロボットを移動先の位置へ移動した後、**ポイントリスト**の右上にある＋ **ポイントを追加**ボタンをクリックし、P1ポイントを保存します。

4. 同じようにP2ポイントを保存します。

5. **プログラミング**画面をクリックし、左側のブロックエリアにある**制御**ブロックグループから**ずっと**ブロックをドラックし、画面中央にある**運転を開始する**ブロックの下にドロップします。

6. **モーション**ブロックグループから**ポイント移動**ブロックをドラックし、**ずっと**ブロックの中にドロップします。ポイントのプルダウンリストをクリックし、P1を選択します。

7. 再度**ポイント移動**ブロックをドラックし、**ずっと**ブロックの中にドロップします。ポイントのプルダウンリストをクリックし、P2を選択します。最終的に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。

プログラミング画面の上にある **保存**をクリックし、プロジェクト名（例えば、"test"）を入力しプロジェクトを保存します。 **実行**ボタンをクリックすれば、ロボットが先にP1ポイントへ移動し、その後P1とP2の間にずっと往復移動を開始します。

# 5 ロボットの基本操作


- 5.1 ユーザーログイン
- 5.2 イネーブル
- 5.3 モード復元
- 5.4 リモートコントロール
- 5.5 手動/自動モード
- 5.6 ジョグ
- 5.7 ドラッグ
- 5.8 緊急停止及び復帰
- 5.9 速度調節
- 5.10 負荷設定
- 5.11 衝突検出設定
- 5.12 アラーム対処

# 5.1 ユーザーログイン

ロボットのユーザーごとに異なる機能を割り当てて、操作権限を管理することができます。

DobotStudio Pro は、ロボットに接続した後、デフォルト機能 (変更可能) に自動的にログインします。デフォルト機能にパスワードがない場合、ユーザーに知らせません。それ以外の場合、パスワードを求めるログイン ウィンドウが表示されます。他の役割権限にログインしたい場合は、この画面で役割権限を切り替えることもできます。


DobotStudio Pro は4つの機能を対応しており、機能ごとに操作権限が異なります（変更可能）。

- 管理者：デフォルトですべての機能の操作権限があります。デフォルトのパスワード: 888888。
- 技術者: ロボットが工場出荷時のデフォルトのログイン時の権限です。デフォルトでは、詳細設定 (設置およびセキュリティ関連の設定) を除くすべての機能を操作する権限があります。デフォルトではパスワードはありません。
- 操作者: デフォルトでは、ジョギングやプロジェクト操作などの基本的なロボット操作に関連する権限のみがあります。デフォルトではパスワードはありません。
- カスタム権限：管理者が作成した名前や権限をカスタマイズで設定した権限です。作成されていない場合は表示されません。

現在のログイン権限に対する操作権限のないUIコントロール（例えば、**編集**ボタンなど）は、クリックすると、下図のように、使用可能な操作権限がないことを示すプロンプトが表示されます。


操作権限のある権限を切り替える必要がある場合は、メイン画面に戻り、情報パネルで機能を切り替えてください。パスワードありの機能に切り替える場合はパスワードの入力が必要ですが、パスワードなしの機能に切り替える場合は直接切り替えることができます。


管理者としてログインすると、デフォルトのログイン機能の設定、カスタム機能の管理、各機能の権限やパスワードの表示・変更などを設定することができます。詳細はユーザー管理をご参照ください。

# 5.2 イネーブル

ロボットは電源投入後、デフォルトではディセーブル状態になります。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行うには、ロボットをイネーブルする必要があります。

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、負荷設定のポップアップウィンドウが表示されます。


ドロップダウンボックスを使用して、負荷のパラメータセットを選択することができます。事前に負荷パラメータ画面でパラメータセットを設定すれば、ここで選択することができます。「カスタム」を選択すると、下記のパラメータを手動で変更することができます:

- **X/Y/Z方向の中心オフセット距離**：エンド負荷の重心から各方向のオフセット距離、各座標軸の方向は画面上のイメージ図をご参照ください。
- **負荷重量**：エンド治具の重量とワークの重量の合計は、ロボットの許容最大荷重を超えないようにしてください。
- **負荷異常検出を実行する**：このオプションをチェックすると、ロボットは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。

⚠ **注意:**
ロボットをイネーブルする際には、**負荷異常検知を実行**にチェックを入れることを推奨します。これにより、安全な運転と機器の保護が確保されます。

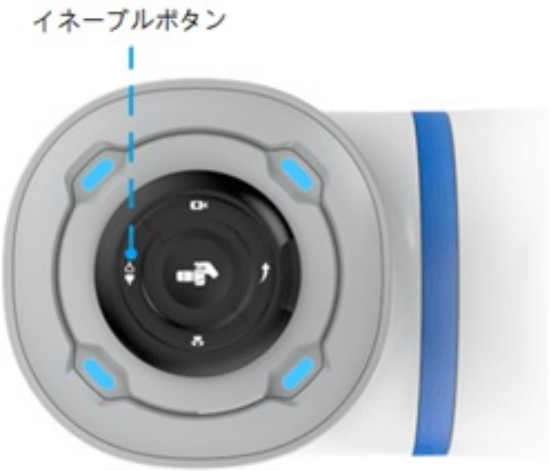
イネーブルが成功すると、イネーブルボタンは**ON**になり、もう一度クリックするとディセーブルになります。


## ロボットイネーブルボタン

**ロボットイネーブルボタンの位置**

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

> **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。




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

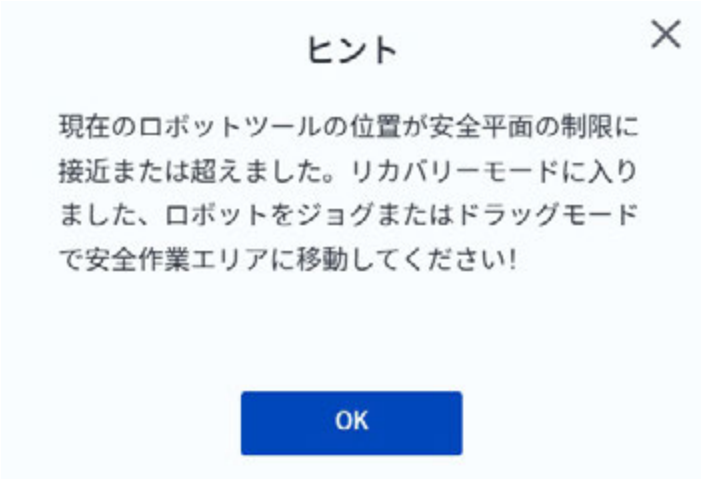

**ロボットイネーブルボタンの操作**

- **イネーブル操作**: ロボットが通電しているがディセーブルの状態（青色インジケーターライトが点灯）で、イネーブルボタンを約1.5秒間長押しすると、紫色のインジケーターライトが点滅します。ボタンを離すと、イネーブル状態に入ります（インジケーターライトが緑色に点灯）。
- **ディセーブル操作**: イネーブル状態でイネーブルボタンを約1.5秒間長押しすると、紫色のインジケーターライトが点滅します。ボタンを離すと、イネーブルが解除され、ロボットがディセーブル状態になります。

**説明:**
ロボットをボタンでイネーブルした後、ロボットは前回イネーブル時の負荷設定を引き継ぎます。ロボットが初めてイネーブルされる場合、負荷パラメータの値はすべて0になります。

# 5.3 モード復元

ロボットが安全エリア外でイネーブルされた場合、以下のようなポップアップが表示され、ロボット末端のインジケーターライトがオレンジ色に変わります。この際、ユーザーにモード復元に入ったことを通知します。




復元モードでは、ユーザーは**ジョグ操作**または**ドラッグ操作**を通じてのみロボットを制御できます。ロボットが安全エリア内に入ると、復元モードが自動的に終了し、ポップアップで通知されます。

# 5.4 リモートコントロール


- 5.4.1 デバイスモード
- 5.4.2 IO/Modbus設定スイッチ

# 5.4.1 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、このモードではDobotStudio Proを使用してロボットを操作できます。また、I/OやModbusを介してロボットをリモート制御することも可能です。
- **TCPモード**は、主にTCPをもとにした独自の制御ソフトウェアを開発する場合に使用されます。このモードでは、非常停止以外、DobotStudio Proでロボットを操作することはできませんが、ロボットの状態や関連設定を確認することは可能です。独自の制御ソフトウェアを開発する必要がある場合は、技術サポートに連絡して「TCP_IPリモート制御インターフェースドキュメント（V4）」を入手してください。

ユーザー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。下線付きの文字をクリックすると、デバイスモードを切り替えることができます。

**説明:**

- ロボットが手動/自動モードの場合、デバイスモードを切り替えることはできません。モード設定で手動/自動モードをオフにする必要があります。
- ロボットが動作中または一時停止中の場合、デバイスモードを切り替えることはできません。
- ロボットがTCPモードに切り替わった後、操作が禁止されているボタンをクリックすると、**ロボットは現在TCPモードにあり、操作できません!** というメッセージが表示されます。

# 5.4.2 IO/Modbus設定スイッチ

IO/Modbus設定スイッチは、リモートIO/Modbus入力が有効になるかどうかを制御します。安全性を考慮し、通常はロボットが単一の入力ソースのみで制御されることを推奨します。そのため、DobotStudio Proを使用してロボットを操作する場合は、IO/Modbus設定スイッチをOFFにすることをお勧めします。

メイン画面の情報パネルにある**IO/Modbus設定**スイッチをクリックすることで、ON/OFFを切り替えることができます。


⚠ **注意:**
I/O設定やModbus設定を通じて、I/OやModbusの実行プロジェクトに関する設定を行ってください。

IO/Modbus入力の具体的な有効範囲は、手動/自動モードの影響も受けます。詳細は以下の表を参照してください。

| 操作モード | IO/Modbus設定ON | IO/Modbus設定OFF |
| --- | --- | --- |
| 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 |

**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 | √ | √ |
| アラームクリア | √ | √ |
| ドラッグに入ります | √ | X |
| ドラッグを終了します | √ | X |
| 予備プロジェクトを選択 | X | √ |

**説明:**
手動モードでは、衝突によって一時停止がトリガーされた場合、ドラッグ操作に移行することはできません。

**安全I/O**

| 機能 | 手動モード | 自動モード |
| --- | --- | --- |
| ユーザー緊急停止 | √ | √ |
| 保護停止 | X | √ |
| 保護停止リセット | X | √ |
| リデュースモード | X | √ |

**説明:**

- 手動/自動モードが未起動の場合、または手動モード中に保護停止がトリガーされた場合、ロボットのジョグ操作やドラッグは可能ですが、プログラムの実行や軌跡再現は許可されません。
- 自動モード中に保護停止がトリガーされた場合、ジョグ操作やドラッグ、プログラムの実行、軌跡再現を含むすべての操作が禁止されます。

# 5.5 手動/自動モード

DobotStudio Proは、異なる操作モードを設定することで現場での安全性を向上させることが可能です。モード設定で有効化できます。

> **説明:**
> CR20ロボットは出荷時に**手動モード**がデフォルトで設定されていますが、その他の機種は出荷時に手動/自動モードが無効となっています。

- **手動モード**: このモードは主にロボットのプログラミングやデバッグに使用されます。
- **自動モード**: このモードは主にロボットの本稼働後の自動運転に使用されます。

モード設定で**操作モード切替**機能を有効化すると、メイン画面の情報パネルに**手動/自動モード切替**スイッチが表示され、デフォルトでは**手動**モードに設定されています。

スイッチをクリックすると**自動**モードに切り替えることができます。自動モードのパスワードが設定されている場合は、切り替え前にパスワードを入力する必要があります。

ロボットが動作中の場合、手動モードと自動モードを切り替えることはできません。ロボットが一時停止または停止状態の場合にのみ、モードの切り替えが可能です。

手動/自動モードで許可される操作については、**IO/Modbus設定スイッチ** を参照してください。

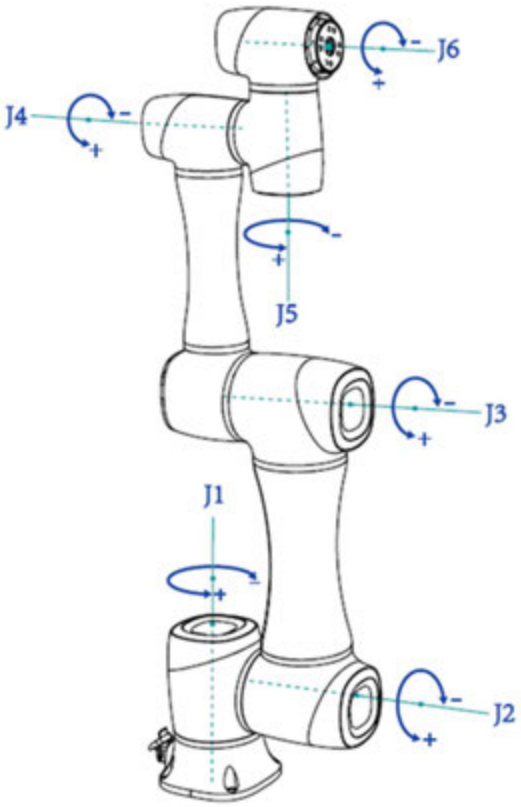
# 5.6 ジョグ

DobotStudio Proを通じてロボットを手動で制御することができます。通常、ポイントのティーチングに使用されます。

ジョグ機能を使用する場合、上部のツールバーの 

ボタンをクリックすると、ジョグ画面が表示されます。この画面では、関節角度またはデカルト座標系に基づいた連続的なジョグまたはインチングが対応します。

# 関節ジョグ


関節ジョグとは、ロボットの1つの関節を回転制御することを指します。

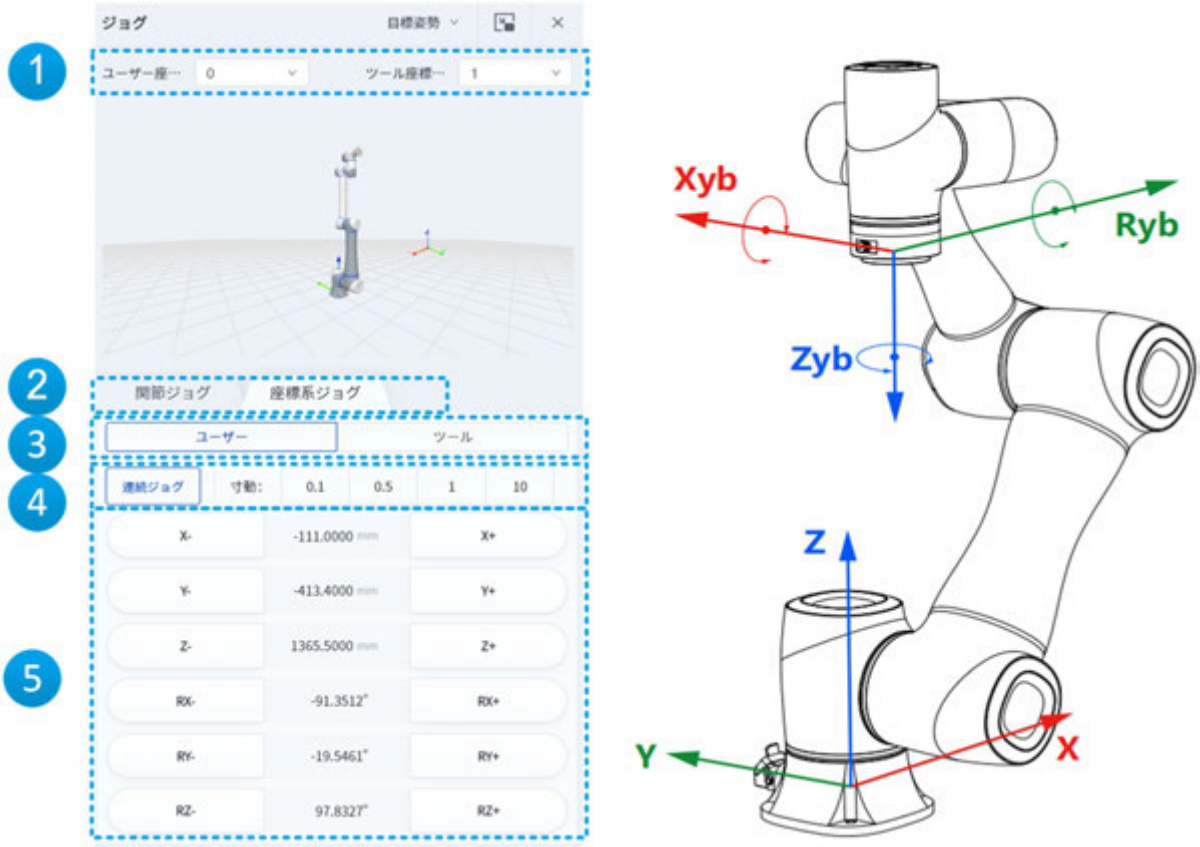
関節ジョグを実行するには、まず上図のエリア①でラベルが付いた**関節ジョグ**を選択し、次にエリア③のモーションボタンを使用して1つの関節の運動を制御します。各関節軸の位置と回転方向は上図右側のイメージ図をご参照ください。

エリア②のオプションは、モーションボタンのモードを設定するために使用されます:

- 連続ジョグ: モーションボタンを押し続けると、ロボットを動し続けます。ボタンを放すと、ロボットが停止します。
- 寸動: モーションボタンをクリックするたびに、ロボットは指定されたインチング距離（単位: °）を移動します。ボタンを押し続けても、1インチングだけが移動されます。

ポイントをティーチングする時には、連続ジョグにより目標点付近までロボットを移動させ、その後寸動により微調整することができます。

# 座標系ジョグ




座標系ジョグとは、ロボットのTCP（Tool Center Point、現在のツール座標系の原点であるツール中心点）を制御して、指定された座標系に沿って平行移動及び回転することを指します。

座標系ジョグを実行するには、まずエリア②でラベルが付いた**座標系ジョグ** を選択し、次にエリア⑤のモーションボタンを使用してTCPの運動を制御します。上図右側のイメージ図は、ユーザー座標系0を例として、各座標軸の定義とその回転方向を示しています。

エリア④のオプションは、モーションボタンのモードを設定するために使用されます:

- 連続ジョグ: モーションボタンを押し続けると、ロボットを動し続けます。ボタンを放すと、ロボットが停止します。

- 寸動: モーションボタンをクリックするたびに、ロボットは指定されたインチング距離（X/Y/Z単位：mm、RX/RY/RZ単位：°）を移動します。ボタンを押し続けても、1インチングだけが移動されます。

ポイントをティーチングする時には、連続ジョグにより目標点付近までロボットを移動させ、その後寸動により微調整することができます。

**モーション基準座標系の切り替え**

エリア①を使用して、ロボットが現在使用しているユーザー座標系とツール座標系を変更することができます。**設定**>**座標系管理**で現在のロボットの座標系を管理することができます。

エリア③は、ロボットがジョグ運転するときの基準座標系を設定するために使用されます:

- ユーザー: ジョグ運転中、ロボットTCPは現在のユーザー座標系に沿って移動します。例えば、X+は、ロボットTCPが現在のユーザー座標系のX軸の正方向に沿って移動することを意味します。
- ツール: ジョグ運転中、ロボットTCPは現在のツール座標系に沿って移動します。例えば、X+ は、ロボットTCPが現在のツール座標系のX軸の正方向に沿って移動することを意味します。。

現在有効な座標系がロボットの3Dモデル上に表示されます。

# ジョグ値の編集

関節ジョグまたは座標系ジョグの運動制御エリアで、任意の数値をダブルクリックすると、**ジョグ値の編集**ダイアログボックスが表示されます。

直接目標位置の関節角度または座標系の値を入力すると、青色のアウトラインシャドウが目標位置の形状を表示します。**移動先**を長押しすると、ロボットが目標位置へ移動します。

# 目標姿勢

目標姿勢を使用すると、ロボットを特定の位置に迅速に移動させることができます。

**安全原点**


**目標姿勢 > 安全原点**をクリックすると、以下のようなダイアログボックスが表示されます。**目標位置へ移動**を長押しすると、ロボットが安全原点へ移動します。ボタンを離すと、ロボットは移動を停止します。


**Z軸の整列**

**目標姿勢 > Z軸の整列**をクリックすると、以下のようなダイアログボックスが表示されます。**目標位置へ移動**を長押しすると、ロボットが現在のツール座標系のZ軸を、現在のユーザー座標系のZ軸に対して最小の姿勢回転角度で自動的に整列させます。ボタンを離すと、ロボットは移動を停止します。

### 梱包姿勢

**目標姿勢 > 梱包姿勢**をクリックすると、以下のようなダイアログボックスが表示されます。**目標位置へ移動**を長押しすると、ロボットが梱包姿勢へ移動します。ボタンを離すと、ロボットは移動を停止します。


### カスタム

**目標姿勢 > カスタム**をクリックすると、操作画面はジョグ値の編集の説明を参照してください。

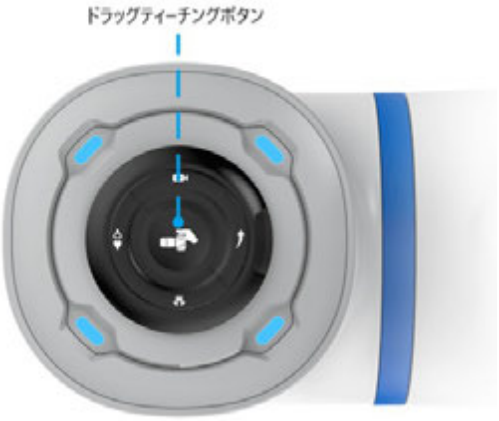
# 5.7 ドラッグ

先端のドラッグティーチング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、位置のティーチング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できます。

> **説明:**
> ロボットをドラッグ操作する前に、負荷パラメータおよび設置角度が正しく設定されていることを確認してください。

- CRAシリーズ（CR20Aを除く）のロボットのドラッグティーチングボタンの位置は下図の通りです。




ロボットのイネーブル状態で（表示灯が緑色に点灯）、ドラッグティーチングボタンを長押しすると (1.5秒以上)、ロボットはドラッグモードに入ります（表示灯が緑色で素早く点滅）。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。

ドラッグモードの状態で、ドラッグティーチングボタンを短押しすると（1.5秒未満）、ドラッグモードを終了することができます。

- CR20Aロボットのドラッグティーチングボタンは末端のフランジの側面にあり、シンボルマークは**FREE**となります。


ロボットのイネーブル状態で（表示灯が緑色に点灯）、このボタンを長押しするとドラッグモードに入

ります（表示灯が緑色で素早く点滅）。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。

ドラッグティーチングボタンから手を離すと、ドラッグモードを終了することができます。

- Magician E6ロボットのドラッグティーチングボタンの位置は下図の通りです。


ロボットのイネーブル状態で（表示灯が緑色に点灯）、ドラッグティーチンボタンを長押しると (1.5秒以上)、ロボットはドラッグモードに入ります（表示灯が緑色で素早く点滅）。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。

ドラッグモード状態で、ドラッグティーチングボタンを短押しすると（1.5秒未満）、ドラッグモードと軌跡記録モードを終了することができます。

# 5.8 緊急停止及び復帰

ロボットが運転中に、突発的な状況が発生した場合、緊急停止機能を使用してロボットを緊急停止にすることができます。

ユーザは以下の方法で緊急停止機能を起動されることができます:

- ロボットコントローラーに接続された緊急停止ボタンを押します。
- 安全IOのユーザ緊急停止信号を入力します。
- ソフトウェア画面の右上にある緊急停止ボタンをクリックします。

緊急停止がトリガーされた後、ロボットには以下の2つの停止状態があります:

- 500ms以内に停止した場合、ロボットはディセーブルされるだけで、電源はオフになりません。
- 500ms経過後もロボットが動作を続けている場合、ロボットは強制的にディセーブルされ、電源もオフになります。

> ⚠ **注意:**
> Magician E6シリーズのロボットでは、緊急停止がトリガーされると直接的に電源がオフになります。

2つの状態に応じて、それぞれ対応するアラームが発生します。

緊急停止が発生すると、緊急停止ボタンのアイコンが点滅状態になります。ロボットを再度イネーブルするには、まず緊急停止ボタンを再度押してリセットを行い、その後アラームをクリアしてからロボットをイネーブルしてください（ロボットの電源がオフになった場合は、ポップアップメッセージに従って電源を入れる必要があります）。

> ⚠ **注意:**
> - ソフトウェア画面上の緊急停止ボタンは、ハードウェアの緊急停止機能を補完するものです。緊急時には、ハードウェアの緊急停止機能を優先してロボットを停止してください。
> - 物理的な緊急停止ボタンや安全IOによる緊急停止も、このボタンの状態を変更しますが、その場合、ソフトウェア画面上のリセットボタンではリセットできません。対応する入力ソースでリセットを行う必要があります。

# 5.9 速度調節

トップツールバー右側の速度スライダーを使用してロボットの速度を調整できます。例えば、プログラムのデバッグ中にロボットの速度を下げることで、操作の安全性を確保します。

# 5.10 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。

以下の方法でロボットの負荷パラメータを設定することができます：

- ロボットをソフトウェアでイネーブルにする際、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。詳細はイネーブルの操作説明をご参照ください。


- プログラムを編集する際、対応するブロックまたはスクリプトコマンドを使用して、プロジェクトの実行時に負荷パラメータを設定することができます。詳細は付録のコマンド説明書をご参照ください。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となります。

  **ブロック**：


  **スクリプト**：

```
SetPayload(payload, {x, y, z}) -- 負荷パラメータをカスタマイズで設定する
```

```
SetPayload(name) -- 事前に設定した負荷パラメータグループを使用する
```

事前に設定した負荷パラメータグループは負荷パラメータ画面で管理します。

# 5.11 衝突検出設定

ロボットが動作中に衝突を検知すると、自動的に停止します。ユーザーは衝突検知の感度や衝突後の具体的な処理方法を設定できます。CRAシリーズのロボットに関する衝突検知の設定については、安全制限を参照してください。Magician E6シリーズのロボットに関する衝突検知の設定については、衝突検知を参照してください。

また、ユーザーは対応するブロックやスクリプトコマンドを使用して、プロジェクトの実行中に衝突検知の感度を設定することも可能です。この方法で設定された衝突検知パラメータは、プロジェクトの実行中のみ有効です。ブロックプログラミングについては、付録の衝突検知機能の設定、衝突後の戻り距離の設定コマンド説明ドキュメントを参照してください。スクリプトプログラミングについては、付録のSetCollisionLevel、SetBackDistanceコマンド説明ドキュメントを参照してください。

# 5.12 アラーム対処

ジョグ操作やポイント保存の方法が正しくない場合、またはロボットの使用方法が不適切な場合（例えば、ロボットアームがリミットに達する、または特異点に位置する場合）、アラームがトリガーされます。ロボットアーム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数字付きの赤い点が表示され、現在のアラーム数を示します。


ロボットにアラームが発生すると、現在のアラームページにアラームレベルアイコン、発生時刻、エラーコード、アラームタイプ、説明、および解決策が表示されます。説明と解決策を参考にしてアラーム問題を解決してください。

ロボットのアラームレベルアイコンの意味は以下の通りです:

| アラームアイコン | アラームレベル | アラームカテゴリ | 対応措置 |
|---|---|---|---|
|  | 0 | 故障 | エラー、動作一時停止、ディセーブルおよび電源オフ |
|  | 1 | 異常 | エラー、動作一時停止およびディセーブル |
|  | 5 | エラー | エラーおよび動作一時停止 |
|  | 10 | 故障 | エラー、動作停止、ディセーブルおよび電源オフ |
|  | 11 | 異常 | エラー、動作停止およびディセーブル |
|  | 15 | エラー | エラーおよび動作停止 |

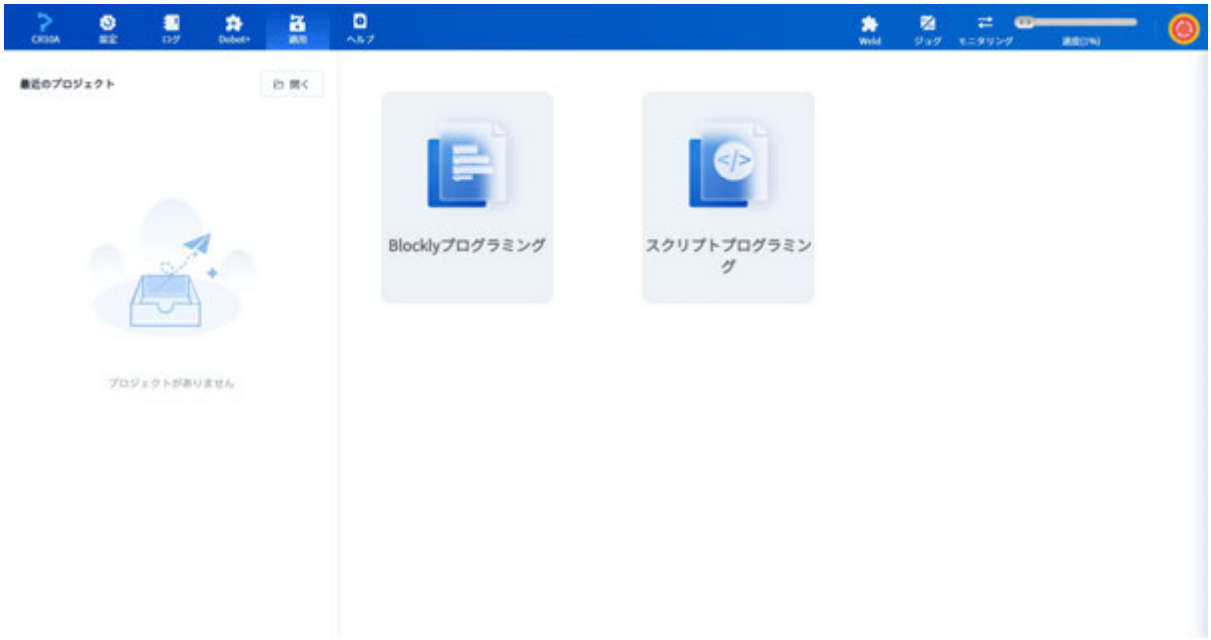
**アラームクリア** をクリックすると、現在のアラームをクリアできます。ただし、一部のアラームは直接クリアできない場合があります。詳細はアラームの解決策を参照してください。

# 6 応用


- 6.1 応用選択
- 6.2 ブロックプログラミング
- 6.3 スクリプトプログラミング
- 6.4 Pythonプログラミング（Magician E6）
- 6.5 デバッグと実行（ブロック/スクリプトプログラミング）
- 6.6 軌跡復元

# 6.1 応用選択

**適用**画面にて適切な方法でプロジェクトを新規作成することができ、ロボットを制御することができます。




- **Blocklyプログラミング**：グラフィカルなプログラミング方法で、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。**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。

-  ：Blockly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では、Blocklyをその形状に従ってサイドバーからプログラミングエリアにドラッグし、組み合わせてプログラミングすることができます。コードを書く必要がありません。

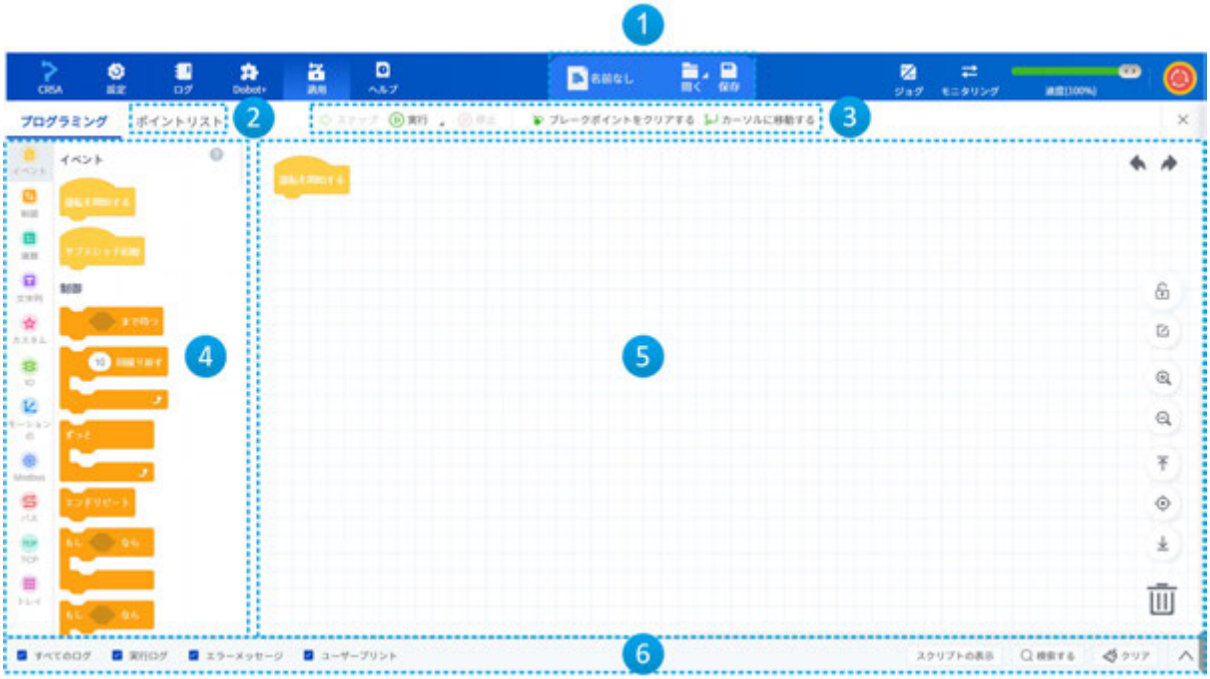
**プロジェクト**: ブロックプログラミングはプロジェクト単位で編集および実行され、デバッグが可能です。アプリケーションのホームページからプログラミングページにアクセスすると、デフォルトで名前のない新規プロジェクトが作成されます。プログラムを作成した後、名前を付けて保存しないとデバッグや実行ができません。プロジェクトはロボットアームのコントローラーに保存され、インポートやエクスポートが可能です。

各プロジェクトには1つのメインスレッドと最大4つのサブスレッドを含めることができます。

- **メインスレッド**: メインスレッドは**実行開始**ブロックを起点とします。メインスレッドではすべてのブロックが使用可能です。
- **サブスレッド**: サブスレッドは**サブスレッド開始**ブロックを起点とします。サブスレッドはメインスレッドと並行して実行され、IOや変数の設定などに使用されますが、動作ブロックは使用できません。

**ポイント保存**: プログラミング中に、ジョグ操作またはドラッグ操作でロボットを移動させ、その位置をポイントリストに保存してティーチングポイントとして使用できます。ポイントリスト内のティーチングポイントはプロジェクトにリンクされ、コマンドのパラメータとして使用できます。複数のプロジェクトで使用可能なティーチングポイントを保存したい場合は、グローバル変数を使用してください。

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っています。




| 番号 | 説明 |
| --- | --- |
| 1 | プロジェクト名の表示とプロジェクト管理に使用されます。 |
| 2 | クリックするとポイント管理画面が開き、プロジェクト内のポイントを管理できます。 |
| 3 | プロジェクトのデバッグおよび実行に関連するボタンです。 |
| 4 | プログラミングに必要なブロックを提供し、分類や色で必要なブロックを検索できます。右上の ? をクリックすると、ブロックの具体的な説明を確認できます。 |
| 5 | ブロックプログラミングのキャンバスであり、ブロックプロジェクトの編集エリアです。<br>編集されて保存されていないブロックには左端に アイコンが表示され、ブロックが変更されたことを通知します。 |
| 6 | 実行ログエリアで、プロジェクトの実行ログを確認できます。<br>• 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ます。<br>• 左側のオプションで表示するログの種類をフィルタリングできます。<br>• **スクリプトを見る**をクリックすると、現在のブロックプログラムのスクリプトを確認できます。<br>• **検索** をクリックすると、ログ内で指定した文字を検索できます。<br>• **クリア** をクリックすると、ログ表示エリアをクリアできます。 |

キャンバス右側のアイコンの意味は下記の通りです。

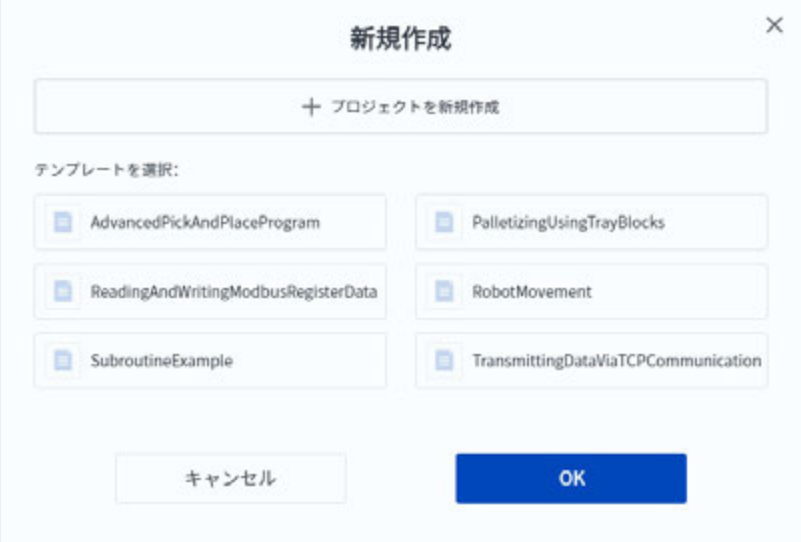
| アイコン | 説明 |
| --- | --- |
|  | プログラミング操作の元に戻し/やり直しに使用されます。 |
|  | プログラミングエリアのロック/アンロックに使用されます。 |
|  | 編集モードに入るために使用されます。詳細は以下のプログラミング説明をご参照ください。 |
|  | プログラムエリアを拡大/縮小するために使用されます。 |
|  | ブロックを上に戻す/中央揃え表示/ブロックを下に戻すの操作に使用されます。 |
|  | プログラムエリアのブロックをここまでドラッグして削除することができます。ブロックを右クリックして削除を選択することもできます。 |

## 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、デフォルトで新規の空白プロジェクトページが表示され、プロジェクト名は**未命名**と表示されます。

をクリックするとファイルメニューが開き、新規作成、開く、名前を付けて保存、インポート、エクスポートの各操作が可能です。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をスクリプトプロジェクトに変換すると、スクリプトプログラミングで開いて編集することができます。

**説明:**
新規プロジェクトを作成する際に、空白プロジェクトを選択するか、ブロックプログラミングのテンプレートを選択できます。




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。

下記の場合では、DobotStudio Proはプロジェクトを自動的にバックアップします。

- DobotStudio Proは、現在開いているプロジェクトが変更されているかどうかを10分間ごとにチェックし、変更されている場合は、現在のプロジェクトをコントローラーに自動的にバックアップします。
- プロジェクトの実行を開始する前に、DobotStudio Proは現在開かれているプロジェクトが変更されているかどうかを確認し、変更されている場合は、現在のプロジェクトをコントローラーに自動的にバックアップします。
- コントローラーから切断されると（アクティブな切断または異常な切断）、DobotStudio Proは、現在開いているプロジェクトが変更されているかどうかを確認し、変更されている場合は、現在のプロジェクトをローカルコンピューター（PC端末またはタブレット端末）に自動的にバックアップします。

コントローラーおよびローカルにバックアップされたプロジェクトは、メニューの **バックアッププロジェクトを開く** オプションから開くことができます。

# 6.2.3 ポイント保存

ユーザーはジョグまたはドラッグでロボットを移動したいポーズに移動させ、その位置をポイントリストに保存することができます。

> **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**ボタンを押しても、ティーチングポイントを追加することができます。


| 番号 | 説明 |
| --- | --- |
| 1 | • 単击 **ポイント追加** ボタンをクリックすると、ロボットの現在の姿勢を新しいティーチングポイントとして保存できます。<br>• ティーチングポイントを選択後に **上書き** ボタンをクリックすると、ロボットの現在の姿勢でそのポイントを上書きできます。<br>• ティーチングポイントを選択後に **削除** ボタンをクリックすると、そのティーチングポイントを削除できます。<br>• **別名フィルター** をクリックすると、別名でフィルターをかけ、条件に一致するポイントをリストに表示できます。 |
| 2 | ポイントリスト。ティーチングポイントを選択後、**NO.** や **ポイント** 以外の任意の値をクリックすると、値を直接変更できます。 |
| 3 | ロボットを指定されたモードで現在選択中のポイントへ移動させます。 |

ポイントリストで選択したポイント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

プログラム編集を開始する前に、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、ロボットの稼働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**制御**ブロックグループから**ずっと**ブロックをキャンバス中の**運転を開始する**ブロックの下にドラッグします。
2. **モーション**ブロックグループ中の**ポイントまで移動**ブロックを**ずっと**ブロックにドラッグし、ポイントの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. もう一つの**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、ポイントの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対応しています:


- **コピー**: キャンバス上に配置したブロックを右クリック（モバイル端末の場合は長押し、以下同様）するとメニューが表示されます。選択したブロックおよびその下に接続されたすべてのブロックをコピーするか、選択した単一のブロックをコピーすることができます（選択したブロックに他のブロックがネストされている場合、それも一緒にコピーされます）。
- **ブレークポイントの追加/削除**: ブレークポイントの設定をサポートするブロックでは、右クリックして**ブレークポイントを追加**を選択できます。ブレークポイントのマーク  は以下の図のように表示されます。この場合、**ブレークポイントを追加**ボタンは**ブレークポイントを削除**に変わります。プログラムがブレークポイントを追加したブロックに到達すると、実行が一時停止します。この際、カーソルはブレークポイントが設定されたブロック上に停止し、そのブロックは実行されません。すでにブレークポイントが追加されているブロックでは、右クリックして**ブレークポイントを削除**を選択すると、そのブロックのブレークポイントを削除できます。


**説明:**

- プログラムが一時停止中の場合、ブレークポイントの追加や削除が可能です。
- ブロックプログラミングでは、**実行開始**および**サブプログラム**以下のブロックのみがブレークポイントの追加をサポートしています。
- サブプログラム全体にブレークポイントを追加した場合（サブプログラム内のブロックにブレークポイントが追加されていない場合）、サブプログラムブロックは以下のように表示されます:


- サブプログラム内の他のブロックにもブレークポイントを追加した場合、サブプログラムブロックは以下のように表示されます:


- ブレークポイントが設定されたブロックをメニューに戻したり、**実行開始**や**サブプログラム**以外に移動した場合、ブレークポイント機能は自動的に無効になります。

- **無効化**: ブロックを選択し、右クリックして**無効化**を選択すると、選択したブロックが無効化され、グレー表示になります。同時に、**スクリプトを見る**ボタンで無効化されたブロックのスクリプトがコメントとして表示されます。無効化されたブロックにはブレークポイントを追加できず、既に追加されているブレークポイントは無効化時に削除されます。
- **削除**: ブロックは以下の3つの方法で削除できます。
  - ブロックを左側のブロックメニューにドラッグして離します。

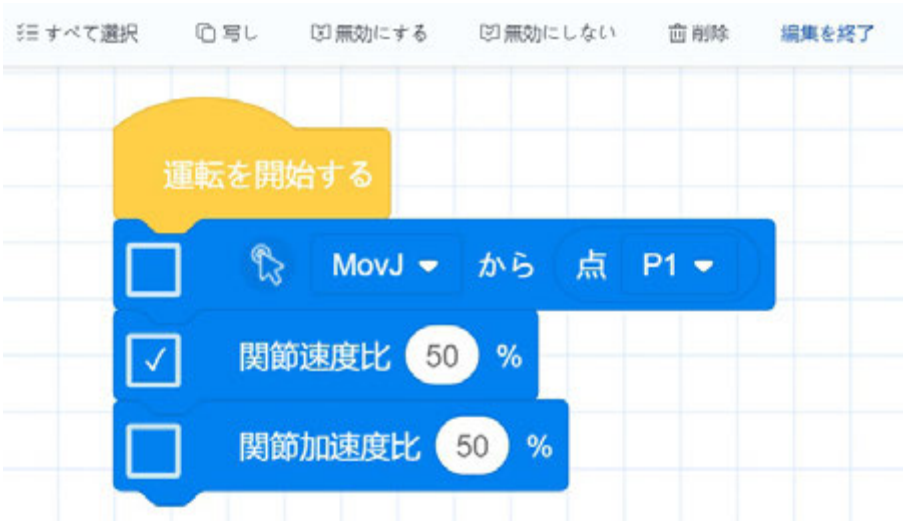
    - ブロックを画面右下のゴミ箱アイコンにドラッグして離します。
    - ブロックを右クリックしてメニューを開き、**削除**を選択します（選択したブロックに他のブロックがネストされている場合、それらも一緒に削除されます）。
- **サブプログラム化**: **実行開始**または**サブスレッド開始**に接続されていないブロック群を右クリックして**サブプログラム化**を選択すると、そのブロック群を1つのサブプログラムに変換できます。サブプログラム化後、1つのサブプログラムブロックとして扱えるため、再利用が容易になり、プログラミング効率が向上します。**サブプログラム編集画面ではこの操作は無効です。**

- ブロックの折りたたみ/展開*: ネストされたブロックを右クリックして、ネスト型ブロックを折りたたむか展開できます。折りたたむとブレークポイントが削除され、折りたたまれたブロックにはブレークポイントを追加できません。

キャンバス右側の  アイコンをクリックすると、編集モードに入ることができます。

編集モードでは、ユーザーは複数選択または全選択を行い、ブロックのコピー、無効化、無効化解除、削除操作が可能です。

**編集終了**をクリックするか、プログラミングエリアで他の操作を行うと、編集モードが終了します。




ブロックに関する詳細な説明はここでは紹介していません。ブロックメニューの右上にある ? をクリックして確認してください。内容は付録Bと同じです。

# 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、Lua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Luaプログラミング環境を提供し、ユーザーは独自のLuaプログラムを作成してロボットの動作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。

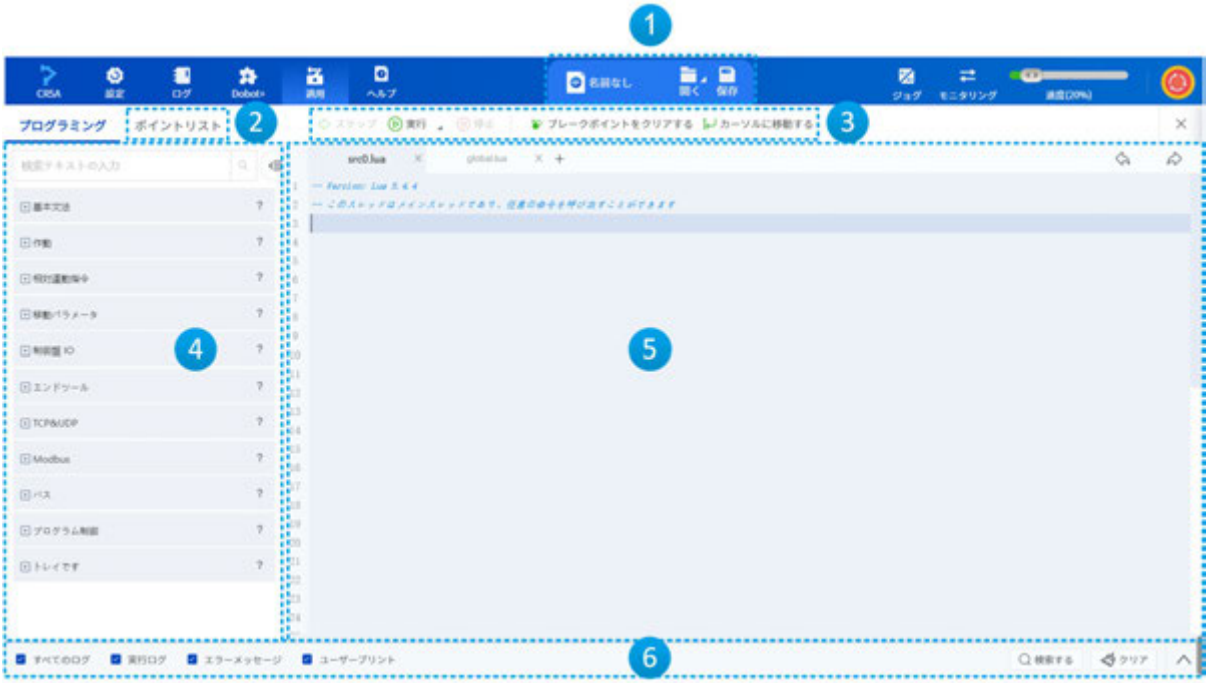
一つのプロジェクトには以下のスクリプトファイルが含まれ、タブ形式で表示されます：

- **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のみに使用されます。
- 0～4つの**サブスレッド**、名前は**src1.lua**~**src4.lua**です。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、モーションコマンドを呼び出すことはできません。

プロジェクトが開始されると、ロボットはメインスレッドとサブスレッドの命令を上から下に順番に実行するため、ユーザーはエントリ（main）関数を定義する必要はありません。 Luaの基本的な構文については、付録Cを参照してください。

**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、ポイントリストを開いて、ロボットの現在のポーズをティーチングポイントとして保存することができます。ティーチングポイントはプロジェクトにリンクされ、コマンドのパラメータとして使用することができます。ティーチングポイントが複数のプロジェクトから呼び出せる場合は、グローバル変数を使用してください。

スクリプトプログラミングのメイン画面は下図のようになります。



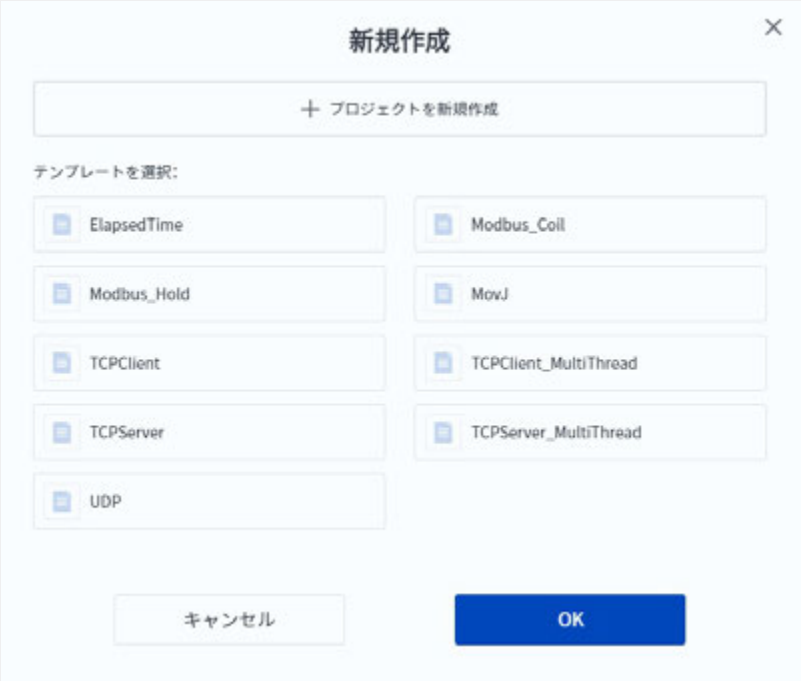

| 番号 | 説明 |
| --- | --- |
| 1 | プロジェクト名の表示とプロジェクト管理に使用されます。 |
| 2 | クリックするとポイント管理画面が開き、プロジェクト内のポイントを管理できます。 |
| 3 | プロジェクトのデバッグおよび実行に関連するボタンです。 |
| 4 | プログラミング関数を検索および使用するために使用されます。 をクリックすると、関数説明ドキュメントを確認できます。 |
| 5 | プログラムの編集エリアで、タブをクリックしてスクリプトファイルを切り替えるか、+ をクリックしてサブスレッドを追加できます。<br>右上の をクリックすると、操作を取り消したり、やり直したりできます。 |
| 6 | 実行ログエリアで、プロジェクトの実行ログを確認できます。<br>• 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ます。<br>• 左側のオプションで表示するログの種類をフィルタリングできます。<br>• **検索** をクリックすると、ログ内で指定した文字を検索できます。<br>• **クリア** をクリックすると、ログ表示エリアをクリアできます。 |

# 6.3.2 プロジェクト管理


スクリプトプログラミング画面を開くと、デフォルトで新規の空白プロジェクトページが表示され、プロジェクト名は**未命名**と表示されます。

をクリックするとファイルメニューが開き、新規作成、開く、名前を付けて保存、インポート、エクスポートの各操作が可能です。

**説明:**
新規プロジェクトを作成する際、空白プロジェクトを選択するか、スクリプトプログラミングのテンプレートを選択できます。



をクリックすると、現在のプロジェクトを保存できます。プロジェクトが未命名の場合、先にプロジェクト名を入力する必要があります。

以下のシナリオで、DobotStudio Proはプロジェクトを自動バックアップします:

- DobotStudio Proは10分ごとに現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをコントローラーに自動バックアップします。
- プロジェクトを実行する前に、DobotStudio Proは現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをコントローラーに自動バックアップします。
- DobotStudio Proがコントローラーとの接続を切断する際（手動で切断する場合や異常切断の場合）、現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをローカル（PCまたはタブレット）に自動バックアップします。

コントローラーおよびローカルにバックアップされたプロジェクトは、 メニューの**バックアッププロジェクトを開く**オプションから開くことができます。

# 6.3.3 ポイント保存

ユーザーは、ジョグ操作またはドラッグ操作を使用してロボットを目的の姿勢に移動させ、ポイントリストにその位置を保存することができます。

> **説明:**
> ポイントリストを開いている間に、CR20Aのエンドフランジ側面の**POINT**ボタンを押すことでもティーチングポイントを追加できます。


| 番号 | 説明 |
| --- | --- |
| 1 | • **ポイント追加**ボタンをクリックすると、ロボットの現在の姿勢を新しいティーチングポイントとして保存できます。<br>• ティーチングポイントを選択後に **上書き**ボタンをクリックすると、ロボットの現在の姿勢でそのポイントを上書きできます。<br>• ティーチングポイントを選択後に **削除**ボタンをクリックすると、そのティーチングポイントを削除できます。<br>• **別名フィルター**をクリックすると、別名でフィルタリングし、条件に一致す |

| | |
|---|---|
| | るポイントをリストに表示できます。 |
| 2 | ティーチングポイントリスト。ティーチングポイントを選択後、**NO.**および**ポイント**以外の任意の値をクリックすると、その値を直接変更できます。 |
| 3 | 指定したモードでロボットを現在選択中のポイントに移動させます。 |

ポイントリストで選択されたポイントに対応する姿勢は、ジョグ操作パネルのシミュレーションエリアに青色の輪郭として表示されます。以下の図のようになります。


# 6.3.4 プログラミング

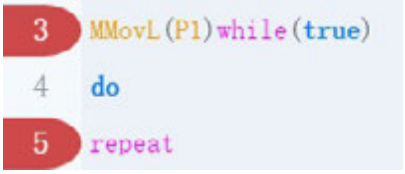
ユーザーは次のような集中的な方法でコマンドを挿入できます:

- 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エリアに追加されます。


- 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コマンドをプログラムエリアにすばやく挿入し、必要に応じてパラメータ値を変更することができます。


- 右側のプログラム編集エリアに直接プログラムを入力できます。また、キーボードのTABキーでコマンドの自動補完をサポートしています。

  **ブレークポイントの追加**: マウスまたは指でコード行をクリックするとブレークポイントを追加できます。ブレークポイントのマークは以下の図の3行目と5行目のように表示されます。スクリプトがブレークポイントに設定されたコード行に到達すると、実行が一時停止します。この時、カーソルはブレークポイント行に留まり、その行のコードは未実行の状態です。



- **ブレークポイントの削除**: 既存のブレークポイントがあるコード行を再度クリックすると、ブレークポイントを削除できます。また、**ブレークポイントのクリア**ボタンをクリックして、現在のプロジェクトのすべてのブレークポイントを削除することも可能です。ただし、プログラム実行中は**ブレークポイントのクリア**ボタンの使用が禁止されています。

> **説明:**
> - プログラムが一時停止している状態で、ブレークポイントの追加や削除が可能です。
> - スクリプトプログラミングでは、`src0.lua` のみがブレークポイントの追加をサポートしています。
> 

プログラムを開始する前に、実現したい機能を明確にしてください。本節では、ロボットを2つのポイント間で繰り返し動作させるプログラムを例として、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で、ロボットの繰り返し動作の始点P1と終点P2を順に追加します。
2. 上述のいずれかの方法で `while` ループ関数を追加してコードを記述します。
3. ループコードの `end` の前に、P1を目標点とする `MovJ` 移動コマンドを追加します。
4. 次に、P2を目標点とする `MovJ` 移動コマンドを追加します。最終的なコードは以下のようになります。

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

これで、簡単循環動作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場合は、プログラム編集エリア上部のタブ右側にある+をクリックしてサブスレッドを追加し、サブスレッドの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さらに詳しい説明はここでは紹介しません。関数メニュー内の をクリックして確認できます。内容は付録Cと同じです。

# 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することができます。

**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。

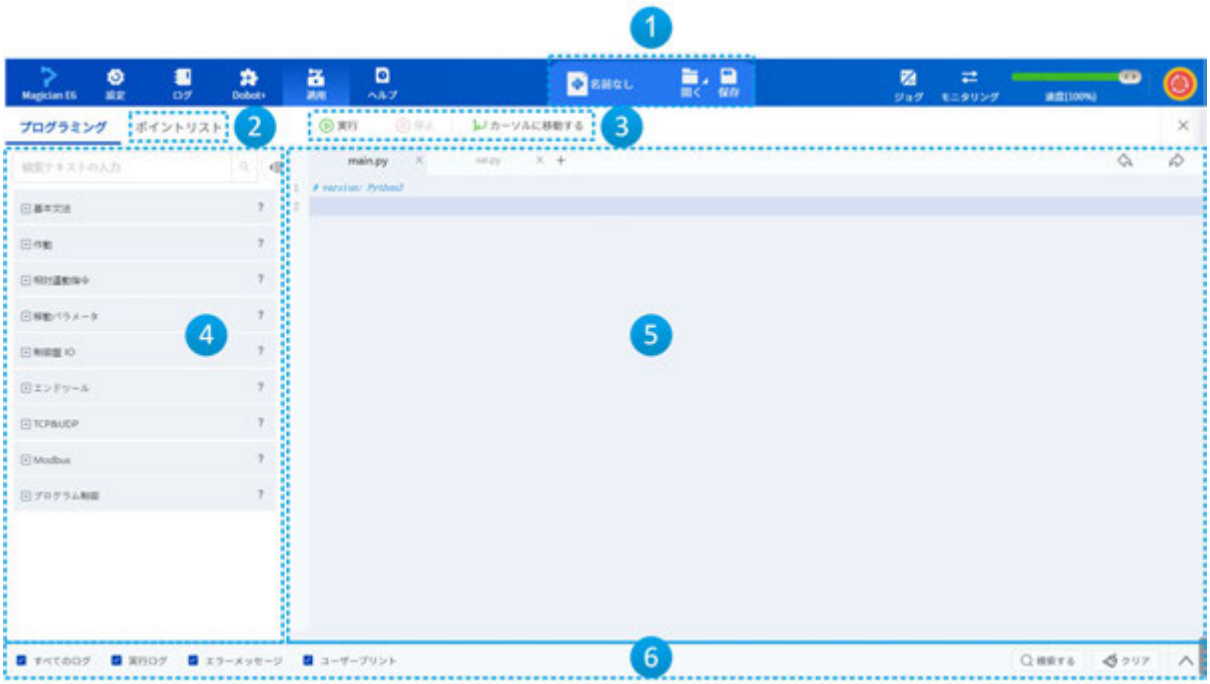
プロジェクトには、タブ形式で表示される下記のPythonファイルが含まれます：

- **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**です。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、モーションコマンドを呼び出すことはできません。

プロジェクトが開始されると、ロボットはメインスレッドとサブスレッドの命令を上から下に順番に実行するため、ユーザーはエントリ（main）関数を定義する必要はありません。 Pythonの基本的な構文については、付録Eを参照してください。

**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、ポイントリストを開いて、ロボットの現在のポーズをティーチングポイントとして保存することができます。ティーチングポイントはプロジェクトにリンクされ、コマンドのパラメータとして使用することができます。ティーチングポイントが複数のプロジェクトから呼び出せる場合は、グローバル変数を使用してください。

Pythonプログラミングのメイン画面は下図のようになります。




| 番号 | 説明 |
| --- | --- |
| 1 | プロジェクト名の表示とプロジェクト管理に使用されます。 |
| 2 | クリックするとポイント管理画面が開き、プロジェクト内のポイントを管理できます。 |
| 3 | プロジェクトの実行に関連するボタンです。 |
| 4 | プログラミング関数を検索および使用するために使用されます。 をクリックすると、関数説明ドキュメントを確認できます。 |
| 5 | プログラムの編集エリアで、タブをクリックしてPythonファイルを切り替えるか、+ をクリックしてサブスレッドを追加できます。<br>右上の をクリックすると、操作を取り消したり、やり直したりできます。 |
| 6 | 実行ログエリアで、プロジェクトの実行ログを確認できます。<br>• 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ます。<br>• 左側のオプションで表示するログの種類をフィルタリングできます。<br>• **検索** をクリックすると、ログ内で指定した文字を検索できます。<br>• **クリア** をクリックすると、ログ表示エリアをクリアできます。 |

## 6.4.2 プロジェクト管理

スクリプトプログラミング画面を開くと、デフォルトで新規の空白プロジェクトページが表示され、プロジェクト名は**未命名**と表示されます。

をクリックするとファイルメニューが開き、新規作成、開く、名前を付けて保存、インポート、エクスポートの各操作が可能です。

をクリックすると現在のプロジェクトを保存できます。プロジェクトが未命名の場合は、まずプロジェクト名を入力する必要があります。

以下のシナリオで、DobotStudio Proはプロジェクトを自動的にバックアップします:

- DobotStudio Proは10分ごとに現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをコントローラーに自動バックアップします。
- プロジェクトを実行する前に、DobotStudio Proは現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをコントローラーに自動バックアップします。
- DobotStudio Proがコントローラーとの接続を切断する際（手動で切断する場合や異常切断の場合）、現在開いているプロジェクトが変更されているか確認し、変更がある場合はプロジェクトをローカル（PCまたはタブレット）に自動バックアップします。

コントローラーおよびローカルにバックアップされたプロジェクトは、メニューの**バックアッププロジェクトを開く**オプションから開くことができます。

# 6.4.3 ポイント保存

ユーザーはジョグまたはドラッグでロボットを移動したいポーズに移動させ、その位置をポイントリストに保存することができます。

> **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**ボタンを押しても、ティーチングポイントを追加することができます。


| 番号 | 説明 |
| --- | --- |
| 1 | • **ポイント追加** ボタンをクリックすると、ロボットの現在の姿勢を新しいティーチングポイントとして保存できます。<br>• ティーチングポイントを選択後に **上書き** ボタンをクリックすると、ロボットの現在の姿勢でそのポイントを上書きできます。<br>• ティーチングポイントを選択後に **削除** ボタンをクリックすると、そのティーチングポイントを削除できます。 |

| | |
|---|---|
| | • **別名フィルター** をクリックすると、別名でフィルタリングして条件に一致するポイントをリストに表示できます。 |
| 2 | ティーチングポイントリスト。ティーチングポイントを選択後、**NO.** や **ポイント** 以外の任意の値をクリックすると、その値を直接変更できます。 |
| 3 | 指定されたモードでロボットを現在選択中のポイントへ移動させます。 |

ポイントリストで選択したポイント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コマンドを挿入できます:

- 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エリアに追加されます。


- 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エリアにすばやく挿入し、必要に応じてパラメータ値を変更することができます。
- 右側のプログラムエリア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。

プログラム編集を開始する前に、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。

1. ポイントリストから、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` 移動コマンドを追加し、目標点をP1に設定します。
4. もう1つの `MovJ` 移動コマンドを追加し、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、プログラムエリア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。

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。内容は付録Eと同じです。

# 6.5 デバッグと実行（ブロック/スクリプトプログラミング）

> ⚠ **注意:**
> プロジェクトのデバッグや実行を開始する前に、ロボットの作業範囲内に人や障害物がないことを確認してください。

ステップ実行、選択行からの実行、選択行からのステップ実行、ブレークポイントのクリアに関する機能は、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のみ適用されます。

以下に、ブロックプログラミングを例にして、デバッグと実行における各アイコンの機能を説明します:


| アイコン | 説明 |
| --- | --- |
| ステップ実行 | 停止状態で **ステップ実行** ボタンをクリックすると、現在のハイライト行を実行完了後、カーソルが次の行または別の行（関数の進入、戻り、gotoなどの場合）に移動し、その時点でスクリプトが一時停止します。これでステップコマンドは終了します。 |
| 実行 | プロジェクトを開き、**実行** ボタンをクリックすると、ワンクリックでプログラムを実行します。この状態では、**ステップ実行**、**ブレークポイントのクリア** ボタンの使用、およびブレークポイントの追加や削除が禁止されます。 |
| 選択行から実行 | • 選択した行からプログラムを実行開始します。<br>• 選択行が無効な場合（空行、コメント行、またはその他の指令のない行など）、ソフトウェアがエラーメッセージを表示し、プログラムは実行されません。 |
| 選択行からステップ実行 | 選択した行から1つの指令を実行し、完了後に停止状態に入ります。 |
| 停止 | クリックすると、プロジェクトの実行を停止します。 |

| | |
|---|---|
| ブレークポイントのクリア | • 現在のプロジェクト内のすべてのブレークポイントをクリアします（ブレークポイントを追加する方法については、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を参照）。<br>• プログラム実行中は、**ブレークポイントのクリア** ボタンの使用が禁止されます。<br>• 新規作成、プロジェクトの開く、または別名保存すると、作業エリアのブレークポイントがクリアされます。 |
| カーソルへ移動 | デバッグまたは実行中に、現在実行中の指令がハイライト表示されます（以下の図はブロックプログラミングの例です）。**カーソルへ移動** ボタンをクリックすると、ハイライト行に素早く移動できます。<br>  |

**説明:**

- スクリプトプログラミングの場合、**選択行から実行**と**選択行からステップ実行**がサポートされるのは、`src0.lua` のみです。選択行とは、入力カーソルがある行であり、青色でハイライトされるプログラム行を指します。
- ブロックプログラミングの場合、**実行開始**以下のブロックのみが**選択行から実行**と**選択行からステップ実行**をサポートします。選択行とは、選択されたブロックが太字で幅広く表示される状態を指します（モバイル端末では300ms長押し、PCでは100ms長押しで選択されます）。

**プログラムモニタリング**

ユーザーは、プログラム変数や **カーソルへ移動** 機能を使用して、実行中のプロジェクトをモニタリングできます。

**外部信号での実行（ブロックプログラミング/スクリプトプログラミング）**

ユーザーは、外部信号（IO または Modbus）を使用して指定されたプロジェクトを実行することもできます。ユーザーがプロジェクトを編集している場合、ソフトウェアはバックアップを行うかどうかを確認するポップアップを表示します。バックアップするかしないかを選択した後、ソフトウェアのホーム画面に戻ります。編集中のプロジェクトが実行するプロジェクトと同じ場合、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。本編集内容を実行する場合は、外部信号で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き、編集を完了して保存してから実行してください。

外部信号でトリガーされたプロジェクトが実行中の場合、アプリケーション画面を開いたりロボットに再接続したりすると、ソフトウェアがプロジェクトが実行中であることを通知するポップアップを表示します。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移動して状態を確認および制御することを選択できます。

# 6.6 軌跡復元

ロボットが一時停止状態の際、軌跡復元機能を利用すると、自動的に一時停止ポイントまで移動し、その後元のプログラムを正常に続行できます。

**軌跡復元機能はデフォルトで有効になっており、無効化することはできません。**


**一時停止状態でジョグ操作をサポート**ボタンをオンにすると、ロボットが一時停止をトリガーした後、以下の操作が許可されます：ジョグ操作、ドラッグ操作、手動モードへの切り替え、自動モードへの切り替え、イネーブル、ディセーブルなど。

**注意:**
ロボットがアラーム（緊急停止によるアラームを含む）をトリガーした場合、一時停止状態に入ります。

**説明:**
現在のプログラムが一時停止状態か停止状態かは、メイン画面の**アプリケーション**アイコンのスタイルで確認できます:

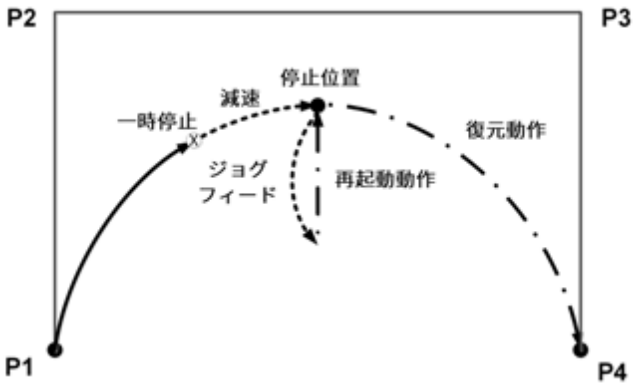
- 

は一時停止状態を示します。
- 

は停止状態を示します。

**軌跡復元機能のオン**

ロボットが一時停止状態のときに軌跡復元機能を有効にすると、ロボットはジョグ速度で一時停止位置にゆっくり移動した後、スクリプト実行速度に戻り、プログラムを再開します。軌跡のずれはありません。実行の効果は以下の図の通りです:



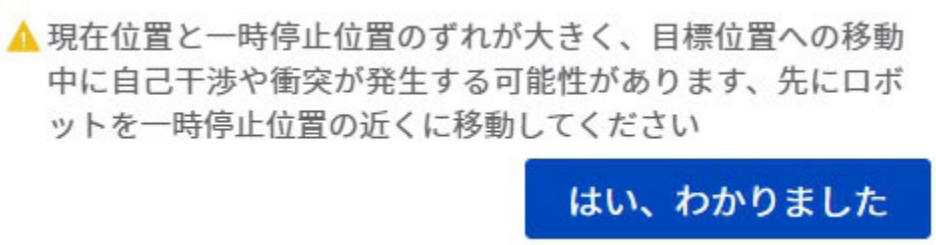

ロボットが一時停止状態の際、プログラミング画面で **続行** ボタンをクリックすると、システムはロボットの現在位置と一時停止状態に入った際の初期位置に偏差があるかどうかを自動的に確認します。

ロボットの現在位置が一時停止時の位置と一致している場合、ロボットは元の軌跡に沿ってプログラムの実行を続行します。

現在位置が一時停止時の位置と異なる場合、以下のようなポップアップ画面が表示されます:


**続行実行**をクリックすると、ロボットは軌跡復元機能のオン設定に応じた動作を続行します。

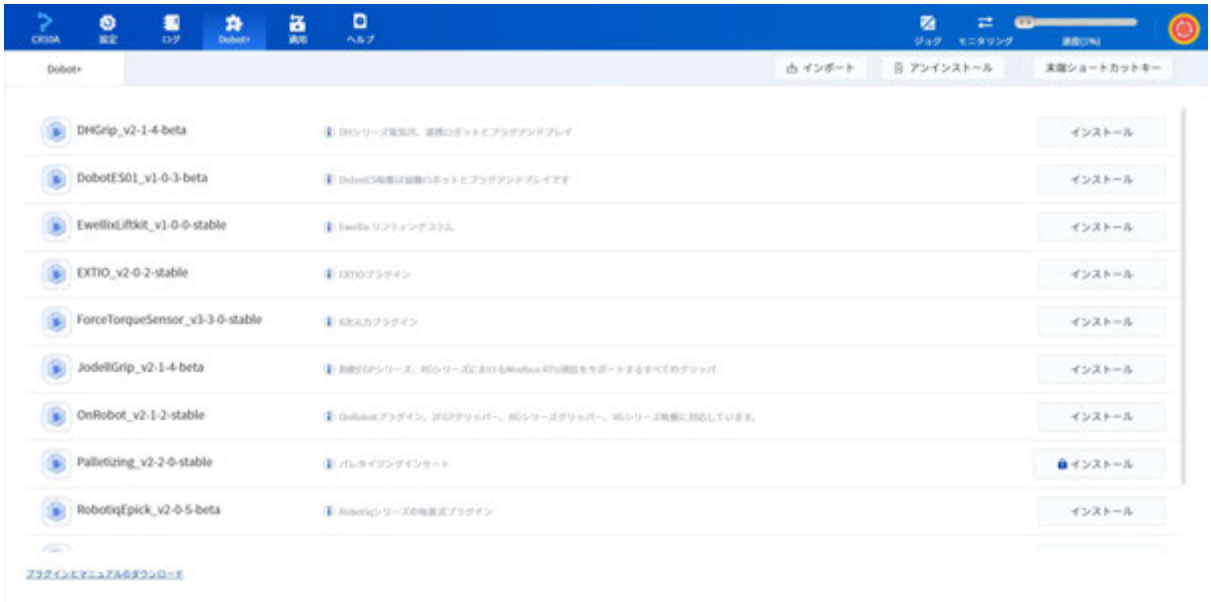
安全性を考慮し、ロボットの現在位置が一時停止位置と大きくずれている場合、目標位置への移動中に自己干渉や衝突などが発生する可能性があります。その場合、以下のような警告ウィンドウが表示されます。




この時、**一時停止位置へ移動**を長押しすると、ロボットは一時停止ポイントまで移動します。

# 7 Dobot+

Dobot+ページは、ユーザーがDobotエコシステムのアクセサリーを迅速に設定および使用できるよう役に立ち、二次開発の手間を省きます。




プラグインの右側にある**「インストール」** ボタンをクリックすると、該当のプラグインをインストールできます。また、上部のタイトルバーにある **「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状態で、タイトルバーにある **「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。

プラグインのインストールが完了すると、**「インストール」** ボタンが**「開く」** ボタンに変わります。クリックすると新しいタブでプラグインが開き、複数のプラグインを同時に開くことが可能です。プラグインの使い方はそれぞれ異なるため、本文では詳細な説明を省略します。

プラグインを追加すると、**ブロックプログラミング**と**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や関数が追加され、プロジェクト実行中にエコシステムアクセサリーを制御できるようになります。

Dobot+ページの右上にある**「エンドショートカット」** をクリックすると、ロボットのエンドボタンにショートカット機能を設定できます。


まず、**「プラグインリスト」** を選択し、その後に**「ショートカット1」**、**「ショートカット2」**、**「長押し」** に対応するプラグイン機能を選択します。ロボットのエンドボタンを押すと、**「ショートカット1」** に対応する機能が実行され、再度エンドボタンを押すと**「ショートカット2」** に対応する機能が実行されます。

Dobot+ページの左下にある**「プラグインおよびマニュアルのダウンロード」** をクリックすると、以下のようなポップアップが表示されます。ウェブページまたはQRコードを使用して、Dobotがサポートするプラグインリストおよび対応するプラグインのユーザーマニュアルを確認およびダウンロードすることができます。

# 8 モニタリング


- 8.1 コントローラーDI/DO
- 8.2 コントローラーAI/AO
- 8.3 末端I/O
- 8.4 安全I/O
- 8.5 Modbus
- 8.6 グルーバル変数
- 8.7 プログラム変数

# 8.1 コントローラーDI/DO

## 8.1.1 DI/DOモニタリング

本画面は、コントローラーのDIとDOの状態と機能を監視・設定するために使用されます。


画面中央のエリアでは、DI/DOの別名と状態を表示・設定することができます。

コントローラーDI/DOはデフォルトで**汎用IO**であり、**システムIO**に設定することもできます。

- **汎用IO**の機能はユーザが定義します。**別名**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。IOの別名機能でコメントをすると、プログラミングを編集する際に、簡単に操作で内容を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度変更することができます。

- **システムIO**とは、下記の**IO設定**画面で機能が設定されたDI/DOです。**別名**は下線なしの青いテキストで表示され、変更することはできません。

DIの右側にある丸い表示灯は、対応するDIの現在の状態を示し、灰色はOFF、緑はONとなります。

DOの右側にあるスイッチは、対応するDOの現在の状態を示し、スイッチをクリックすると、対応するDOの状態をON/OFFを切り替えて制御することができます。**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。

画面の上にある**設定**ボタンをクリックすると、設定画面に移行し、IO設定、プロジェクト選択、詳細設定を行うことができます。

## 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ます。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、監視画面の表示灯はDOと同じようにスイッチになり、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、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たり、プロジェクトを続行したりすることができます。

> **説明:**
> 仮想DIは設定後ずっと有効になり、プロジェクトの実行中にコマンドによって対応するDIを読み込むと仮想値が読み取られ、DIの実際値を取得することはできません。この状態を避けたい場合、プロジェクトを実行する前に、DIを実際のDIに戻してください。

### 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すればDI/DOのリモート制御機能を設定することができます。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が実行されます。詳細はプロジェクト選択をご参照ください。<br>ロボットが一時停止している際、プロジェクト（またはその他の形式のコマンドキュー）の実行を続行します。 |
| 停止 | 実行中のプロジェクト（または他の形式のコマンドキュー）を停止します |
| 一時停止 | 実行中のプロジェクト（または他の形式のコマンドキュー）を一時停止します。 |
| イネーブル | ロボットが電源投入している場合、ロボットをイネーブルにします。 |
| ディセーブル | ロボットがイネーブルしている場合、ロボットをディセーブルにします。 |
| 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。 |
| ドラッグモードに入る | ロボットがイネーブルしている場合、ロボットをドラッグモードにします。 |
| ドラッグモードを終了 | ロボットがドラッグモードの場合、ロボットのドラッグモードを終了します。 |

| DO機能 | 説明 |
| --- | --- |
| 実行状態 | ロボットがプロジェクト、TCPモードコマンドキュー、軌跡再現を実行している場合、1を出力します。それ以外の場合、0を出力します。当該状態は、ロボットが運動中かどうかではなく、ロボットがプロジェクトを実行しているかどうかを示します。 |
| 停止状態 | ロボットが停止している場合、1を出力します。それ以外の場合、0を出力します。 |
| 一時停止状態 | ロボットが一時停止している場合、1を出力します。それ以外の場合、0を出力します。 |
| 安全原点状態 | ロボットが安全原点位置にある場合、1を出力します。それ以外の場合、0を出力します。 |
| セーフスキン一時停止状態 | ロボットがセフティスキン一時停止状態である場合、1を出力します。それ以外の場合、0を出力します。 |
| 待機状態 | ロボットが待機状態（イネーブル中、停止中かつアラームなし）である場合、1を出力します。それ以外の場合、0を出力します。アイドル状態とは、ロボットがいつでもコマンドを受け入れて実行できることを意味します。 |
| 通電状態 | ロボットが電源投入する場合、1を出力します（電源投入完了ではありません）。それ以外の場合、0を出力します。 |
| イネーブル状態 | ロボットがイネーブルしている場合、1を出力します。それ以外の場合、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、1を出力します。それ以外の場合、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、1を出力します。それ以外の場合、0を出力します。 |
| ドラッグ状態 | ロボットがドラッグ状態である場合、1を出力します。それ以外の場合、0を出力します。 |
| 未実行時はLow | プロジェクト、TCPモードのコマンドキューまたは軌跡再現未実行、一時停止または停止している場合、0を出力します。対応するIO状態は、実行中にコマンドによって設定することができます。 |
| 未実行時はHigh | >プロジェクト、TCPモードのコマンドキューまたは軌跡再現未実行、一時停止または停止している場合、1を出力します。対応するIO状態は、実行中にコマンドによって設定することができます。 |
| 異常停止時はLow | 異常停止している場合、1を出力します。対応するIO状態は、実行中にコマンドによって設定することができます。<br>下記の状況で、異常停止が発生する可能性があります:<br>• 安全機能が停止している場合（例えば：衝突検出、安全壁と安全エリア、セフティIOなど）。 |

| | |
|---|---|
| | • アラームが発生している場合。<br>• プロジェクト実行時異常が発生している場合。 |

このうち、**停止時はLow/停止時はHigh/異常停止時はLow**の3つの状態は他の状態と異なり、条件を満たした場合のみ指定した状態を出力します。プロジェクトまたはTCPモードでの動作中にコマンドによりステータスを自由に変更できるため、外部機器とロボットの連携を1つのDOで素早く制御でき、判定ロジックが簡素化することができます。

**説明:**
- **未実行**: プログラムが開始されていない状態を指します。未実行時は、IO状態がロックされ、変更できません。
- **異常停止**: プログラムが構文エラー、衝突停止、緊急停止などの異常な状況により停止した状態を指します。異常停止時にはIO状態はロックされません。

**例:**
**未実行時Low**を例としてこのタイプの状態の使い方を説明します：ディスペンサー応用シーンの場合、プロジェクトでDO1にてディスペンサーを制御します。DO1が1になるとディスペンサーが起動、0になるとディスペンサーが停止します。DO1を**未実行時Low**に設定すれば、プロジェクトが未実行、一時停止、停止している場合、自動でディスペンサーを停止することができます。しかもプロジェクト中のコマンドによるディスペンサーの制御に影響を与えません。

機能が**候補プロジェクト**のDIは、この画面では機能設定を行いません。先にに[プロジェクトを選択]画面でDIをリリースする必要があります。

設定変更後、画面の右上にある**保存**ボタンをクリックして設定を完了してください。

**注意:**
- 全てのリモートトリガ源が同時に有効になりますので、設備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]をご参照ください。
- 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恐れがあります。

**プロジェクト選択**

**デフォルトで実行されるプロジェクト**を選択する場合、**開始**のDIがトリガされると、選択したプロジェクトを直接に実行することができます。


**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が表示されます。

**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をクリアすることができます。

**グループIO選択プロジェクト**を選択した場合、グループIOによって複数のプロジェクトを設定することができます。


1. +または-ボタンをクリックすると、グループIOの割り当てアドレス数を増減することができます。アドレスが多ければ多いほど（最大数4）、設定できるプロジェクトが多いです。
    - 1つのアドレス：2個のプロジェクトを構成可能
    - 2つのアドレス：4個のプロジェクトを構成可能
    - 3つのアドレス：8個のプロジェクトを構成可能
    - 4つのアドレス：16個のプロジェクトを構成可能
2. アドレスはドロップダウン ボックスから変更できます。グループIOに割り当てられたアドレスは、リモートIOまたはセーフティIO（CCBOX）で重複使用することができません。
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することができます。実際の運用に合わせて設定してください。空白のままにしても構いません。

    プロジェクトを実行する前に、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

実行せず、アラーム発生します。

> **例:**
> 上記の図では、DI1～DI4の4つのアドレスを割り当てる例として:
> - 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。

設定変更後、**保存**ボタンをクリックして設定を完了してください。

**高度な設定**


**トリガーモード**

DIの機能をどのようにトリガーするかを設定します。上昇エッジはDIがOFFからONに変わるときに設定された機能をトリガーし、下降エッジはDIがONからOFFに変わるときに設定された機能をトリガーします。

**モード選択**

**CRAシリーズ**では、実際のハードウェア配線モードに基づいてDOのモードを選択する必要があります。詳細は対応するCRAハードウェアマニュアルを参照してください。

設定を変更した後、**保存**ボタンをクリックして設定を完了します。

⚠️ **注意:**
DobotStudio Proソフトウェアのパラメータ設定と**CRAシリーズ**のハードウェア配線を同時に完了する必要があります。これにより、DOモードの設定が成功します。

# 8.2 コントローラーAI/AO

本画面は、コントローラーのAIおよびAOの状態とモードを監視・設定するために使用されます。

> **説明:**
> Magician E6 ロボットに接続する場合、この画面は表示されません。


アナログ入力・出力は、コントローラーのAI・AOインターフェースの実際の値を表示するために使用されます。電圧と電流の2つの検出モードを対応しており、**変更**ボタンをクリックすると、切り替えることができます。また、AOは出力値を手動で変更できます。設定変更後、**変更確認**ボタンをクリックすると、有効になります。

# 8.3 末端I/O

本画面は、ロボットの末端のIOの状態とモードを監視するために使用されます。

> **説明:**
> 下図はCRAシリーズの単一のエンドコネクタを例とします。複数のエンドコネクタを備えた機種の場合（CR20A）、複数のタブが表示されます。
> Magician E6の場合、エンドコネクタにはRS485及びAIインターフェースがなく、2チャンネルのDIと2チャンネルのDOのみ備えます。


### DI/DO

DI/DO別名の列にある または テキストをクリックすると、別名のコメントを編集できます。ユーザーは別名機能を使用してIOの機能に注釈を付けることができ、後続のプログラミングや操作時に確認しやすくなります。

DIの右側にある円形インジケーターは、対応するDIの現在の状態を示します。灰色はOFF、緑色はONを表します。

DOの右側にあるスイッチは、対応するDOの現在の状態を示します。スイッチをクリックするとON/OFFを切り替え、対応するDOの状態を制御できます。

### 仮想DI

右上にある**設定**ボタンをクリックすると、仮想DIを設定することができます。DIが仮想DIに設定される場合、表示灯がスイッチに変わり、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、外部DI機器の入力をシミュレートしたり、DIに関する機能をデバッグしたりすることができます。例えば、プロジェクト実行中にDIに関する判定条件を満たし、プロジェクトを継続実行することができます。


> **説明:**
> 仮想DIは設定後ずっと有効になり、プロジェクトの実行中にコマンドによって対応するDIを読み込むと仮想値が読み取られ、DIの実際値を取得することはできません。この状態を避けたい場合、プロジェクトを実行する前に、DIを実際のDIに戻してください。

### 末端動作モード

- 動作モードが**RS485**の場合、末端IOピン1と2の機能はそれぞれ485Aと485Bになります。
- 動作モードが**アナログ入力**の場合、末端IOピン1と2の機能はそれぞれAI_1とAI_2となり、この画面で入力値をリアルタイムで監視できます。

**説明:**
RS485対応のエンドツールを使用する場合、動作モードを**RS485**に設定する必要があります。

# 8.4 安全I/O

本画面は、安全I/Oの状態の確認と安全I/Oの機能設定を行うために使用されます。


安全I/Oの状態の列の表示灯は、対応するI/Oの状態を示します。緑色はHigh電圧、灰色はLow電圧となります。

安全IOはデュアルチャネル構成となります。デュアルチャネルSIの2つの表示灯はそれぞれ2つのチャネルの状態を示します。トリガー ロジックは構成機能によって異なります。デュアルチャネルSOの2つのチャネル状態は同期され、1つの表示灯のみを使用します。

> **説明**
> Magician E6の安全IOはコントローラーDI/DOの配線端子を共用します。実際運用に応じて、シングルチャネルまたはデュアルチャネルを構成することができます。既にシステムIOとして設定された端子には安全IO機能を設定することができません。逆も同様です。

**設定**ボタンをクリックすると、安全I/Oの機能構成を変更することができます。

設定変更後、**保存**ボタンをクリックして設定を完了してください。


**設定可能なSI機能**

| 機能 | 説明 |
| --- | --- |
| ユーザー緊急停止 | ユーザー緊急停止入力は、ユーザーが使用するための緊急停止インターフェースであり、外部の緊急停止デバイスを接続することができます。<br>ユーザー緊急停止入力は**デフォルトでHigh電圧ノーマルクローズ信号入力で、任意チャンネルがLow電圧になるとロボットが緊急停止状態になります。**当該機能はデフォルトで緊急停止状態を出力します。場合によっては緊急停止がセルフロックになる可能性があります。この状況を避けたい場合は、構成可能なSIを**ユーザー緊急停止（緊急停止状態出力なし）**に変更し、対応するインターフェースをユーザー緊急停止入力に使用してください。 |
| 保護停止 | 保護停止入力は、外部保護機器（安全ドア、セーフティライトカーテンなど）を接続するためのインターフェースです。<br>保護停止入力は**デフォルトでHigh電圧ノーマルクローズ信号入力で、任意チャンネルがLow電圧になるとロボットが保護停止状態（一時停止状態）になります。**<br>• 保護停止リセット入力インタフェースを設定している場合、保護停止状態を解除するには、保護停止入力信号の復旧と保護停止リセットの入力を同時に行う必要があります。その後、ソフトウェアで動作継続を確認してから、ロボット動作を再開することができます。<br>• 保護停止リセット入力インターフェースが設定していない場合、保護停止入力信号を復旧することで保護停止状態が解除され、ロボット動作を再開することができます。<br>SI_3とSI_4には、固定機能として構成されています。 |
| 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イスです。<br>保護停止リセット入力は**デフォルトでHigh電圧ノーマルオープン** |

| | |
|---|---|
| | **信号入力で、デュアルチャネルのエッジ同時に立ち上がると保護停止状態をリセットすることができます。** |
| リデュースドモード | リデュースモード入力は、ユーザーがロボットをリデュースモードに切り替えるためのインターフェースです。デフォルトではHigh電圧の常閉信号入力であり、いずれかの入力がLow電圧になるとロボットがリデュースモードに入ります。 High電圧入力が復帰すると、ロボットはリデュースモードを終了して通常モードに戻ります。ロボットがリデュースモードをトリガーすると、CRAの移動速度および安全制限はリデュースモードの設定値に切り替わります。また、Magician E6のグローバル速度は10%に制限され、変更することはできません。これにより、ロボットの高速動作による安全リスクを回避します。 |

**⚠ 注意**

実際運用時、SI信号の変化間隔が150ms以上にするようにしてください。SI信号のジャンプによりロボットが動作状態異常になる可能性があります。

例えば、保護停止リセット入力が設定していない場合、保護停止入力信号のジャンプ（変化間隔が150ms未満）により、ロボットが一時停止後に自動的に動作を再開しない場合があります。下記の方法にて解決する必要があります：

- ソフトウェアにてプロジェクトを一時停止し、その後プロジェクトの実行を再開します。
- 再度保護停止入力信号を入力し、150ms以上保持します。

**設定可能なSO機能**

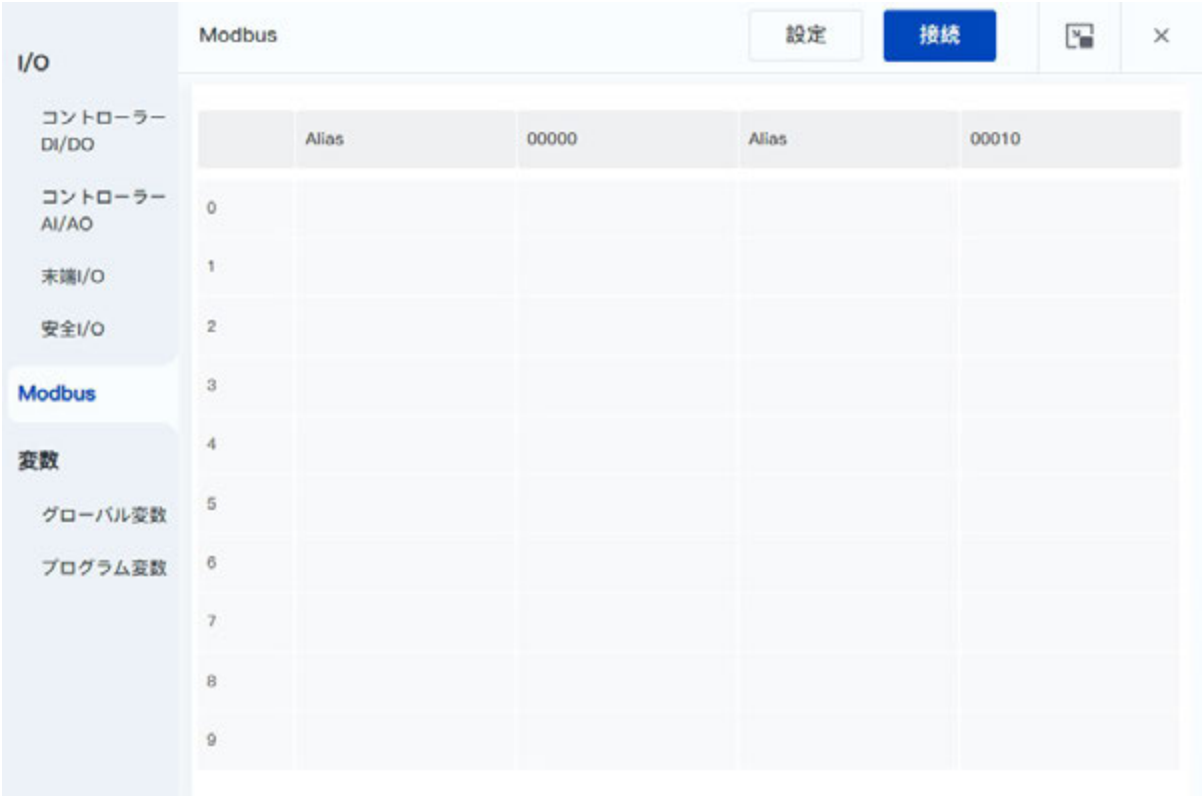
| 機能 | 説明 |
|---|---|
| 緊急停止状態の出力 | ロボットが**緊急停止状態になる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。<br>任意の原因からの緊急停止でも出力されます。<br>SO_1とSO_2には、固定機能として構成されています。 |
| 非停止状態の出力 | ロボットが**自動運転状態時、非停止状態になる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。<br>当該状態の判断基準としては、ロボットが関節運動中かどうかではなく、プロジェクトが実行中かどうかになります。例えば、プロジェクト実行中に、DI信号をONになるまで待つコマンドを実行する最中に、ロボットが運動していないが、非停止状態になり、Low電圧を出力します。プロジェクト一時停止中に、停止状態になり、High電圧を出力します。 |
| リデュースドモードの出力 | ロボットが**リデュースドモードになる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。 |
| 運動中状態の出力 | もしロボットの**1つ以上の関節が1°/秒を超えて動く場合（ドラッグモードを除く）、ロボットが運動中状態になり、Low電圧を出力します。**それ以外の場合、High電圧を出力します。 |

| | |
|---|---|
| 安全原点出力 | ロボットが**安全原点位置にある場合、High電圧を出力します。**それ以外の場合、Low電圧を出力します。<br>安全原点は安全設定にて変更することができます。 |
| 保護停止状態の出力 | ロボットが**保護停止状態になる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。 |
| システム緊急停止状態出力 | ロボットが**システム緊急停止状態になる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。<br>システム緊急停止は緊急停止物理ボタンまたはソフトウェア上の緊急停止ボタンが押されたことによる緊急停止です。 |
| ユーザー緊急停止状態出力 | ロボットが**ユーザー緊急停止状態になる場合、Low電圧を出力します。**それ以外の場合、High電圧を出力します。<br>ユーザー緊急停止は安全IOによる緊急停止です。 |

# 8.5 Modbus

## 8.5.1 Modbusモニタリング

本画面は、PCソフトウェアをModbusマスタ局としてロボットコントローラーに付属するModbusスレーブ局に接続し、レジスタ値の確認や変更を行い、コントローラーがスレーブ局として機能する場合のリモートModbus制御機能を設定するために使用されます。




ページ上部の**接続**をクリックして、接続するスレーブデバイスを設定します。

- **スレーブIP**：Modbusデバイスのアドレス。コントローラー内蔵のModbusスレーブに接続する場合は、コントローラーの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。コントローラー内蔵のModbusスレーブに接続する場合は502を入力します。
- **スレーブID**：スレーブデバイスのID。
- **機能コード**：スレーブデバイス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。コントローラー内蔵のModbusスレーブに接続する場合は、付録A Modbusレジスタ定義を参照してください。
- **スキャン周期**：ロボットアームがスレーブをスキャンする間隔時間。


接続に成功すると、ページ中央の表にスレーブの各アドレスの別名と値が表示されます。別名のセルをダブルクリック（PC版）またはシングルクリック（モバイル版）すると、別名を編集できます。レジスタタイプがコイルレジスタまたは保持レジスタの場合、レジスタ値のセルをダブルクリック（PC版）またはシングルクリック（モバイル版）すると、値を編集できます。

ページ上部の**設定**ボタンをクリックすると設定ページに移動し、コイルレジスタのトリガーモードを設定したり、リモートコントロール機能に対応するアドレスの設定情報を確認したり、リモートスタートのプロジェクトを設定することができます。

## 8.5.2 Modbus設定


**トリガーモード**は、コイルレジスタの機能をどのようにトリガーするかを設定します。上昇エッジはコイルレジスタが0から1に変化したときに設定された機能をトリガーし、下降エッジはコイルレジスタが1から0に変化したときに設定された機能をトリガーします。

**コイルレジスタ**と**接点レジスタ**のアドレス情報は閲覧のみ可能で、変更はできません。それぞれの機能の説明は以下の通りです。

| コイルレジスタ機能 | 説明 |
| --- | --- |
| 開始 | ロボットアームが待機状態のとき、指定されたプロジェクトを実行します（詳細は下記のプロジェクト選択参照）。<br>ロボットアームが一時停止中の場合は、プロジェクト（または他の形式のコマンドキュー）の実行を再開します。 |
| 停止 | 実行中のプロジェクト（または他の形式のコマンドキュー）を停止します。 |
| 一時停止 | 実行中のプロジェクト（または他の形式のコマンドキュー）を一時停止します。 |
| イネーブル | ロボットが通電している状態で、ロボットを有効化します。 |
| ディセーブル | ロボッイネイネーブルされている状態で、ロボットをディセーブルします。 |
| アラーム解除 | ロボットの現在のアラームを解除します。 |

| | |
|---|---|
| ドラッグモードに入る | ロボットがイネーブルされている状態で、ドラッグモードに切り替えます。 |
| ドラッグモードを終了 | ロボットがドラッグモードにある場合、ドラッグモードを終了します。 |

| 接点レジスタ機能 | 説明 |
|---|---|
| 実行状態 | ロボットがプロジェクト、TCPモードコマンドキュー、軌跡再現を実行している場合、1を出力します。それ以外の場合、0を出力します。当該状態は、ロボットが運動中かどうかではなく、ロボットがプロジェクトを実行しているかどうかを示します。 |
| 停止状態 | ロボットが停止している場合、1を出力します。それ以外の場合、0を出力します。 |
| 一時停止状態 | ロボットが一時停止している場合、1を出力します。それ以外の場合、0を出力します。 |
| 安全原点状態 | ロボットが安全原点位置にある場合、1を出力します。それ以外の場合、0を出力します。 |
| セーフスキン一時停止状態 | ロボットがセフティスキン一時停止状態である場合、1を出力します。それ以外の場合、0を出力します。 |
| アイドル状態 | ロボットがアイドル状態（イネーブル中、停止中かつアラームなし）である場合、1を出力します。それ以外の場合、0を出力します。アイドル状態とは、ロボットがいつでもコマンドを受け入れて実行できることを意味します。 |
| 通電状態 | ロボットが電源投入する場合、1を出力します（電源投入完了ではありません）。それ以外の場合、0を出力します。 |
| イネーブル状態 | ロボットがイネーブルしている場合、1を出力します。それ以外の場合、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、1を出力します。それ以外の場合、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、1を出力します。それ以外の場合、0を出力します。 |
| ドラッグ状態 | ロボットがドラッグ状態である場合、1を出力します。それ以外の場合、0を出力します。 |
| リカバリモード状態 | ロボットがリカバリモードである場合、1を出力します。それ以外の場合、0を出力します。 |

設定変更後、**保存**ボタンをクリックして設定を完了してください。

⚠ **注意:**

- 全てのリモートトリガ源が同時に有効になりますので、設備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ご参照ください。
- 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恐れがあります。

**デフォルトで実行されるプロジェクト**を選択する場合、**開始**のコイルがトリガされると、選択したプロジェクトを直接に実行することができます。


**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が表示されます。

**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をクリアすることができます。

**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設定することができます。


+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256）。**開始**のコイルがトリガされると、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。

設定変更後、**保存**をクリックして設定を完了してください。

# 8.6 グローバル変数

本画面は、グローバル変数を設定するために使用されます。グローバル変数はコントローラー内に保持され (停電後も保持できます)、複数のプロジェクトから呼び出すことができます。

グローバル変数を設定した後、**ブロックプログラミング**でグローバル変数に関連するブロックを使用するか、**スクリプトプログラミング**で変数名を直接に呼び出すことができます。


**追加**ボタンをクリックすると、グローバル変数を追加することができます。グローバル変数を選択し、**変更**ボタンをクリックすると、変数のプロパティを変更することができ、**削除** ボタンをクリックすると、選択した変数を削除することができます。


グローバル変数が対応するタイプは以下の通り：

- number：数値。値の範囲の設定を対応しています。プロジェクト内の数値変数に、値の範囲を超える値を代入すると、ロボットはエラーが発生して停止します。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（true/false）
- string：文字列。画面に入力する文字列を二重引用符で囲む必要はありませんが、プロジェクト内の文字列型のグローバル変数の値を変更する場合は、文字列を二重引用符で囲む必要があります。
- table：テーブル（配列を含む）。ポイントまたはカスタマイズ形式への設定を対応します。
    - **ポイント**形式に設定する場合、保存したいポイントにロボットを移動した後、**ポイント取得**をクリックします。


- **カスタム**形式に設定する場合、変数の値を入力する必要があります。グローバル変数ページでの入力および表示する変数値の形式は、JSONデータ形式の制約に従う必要があり、プロジェクト内で使用されるデータ形式とは異なります。具体的なルールは以下の通りです:
  - 配列について：グローバル変数ページでは、`{}`の代わりに`[]`を使用します。例えば、プロジェクト内での形式が`{1,2,3}`の配列の場合、グローバル変数ページでは`[1,2,3]`の形式で入力および表示します。
- `{key=value}`形式のテーブルについて：グローバル変数ページでは、`{"key":value}`形式に変更します。例えば、プロジェクト内での形式が`{a=1, b="test"}`の場合、グローバル変数ページでは`{"a":1, "b":"test"}`の形式で入力および表示します。


設定完了後、**追加**をクリックしてグローバル変数を追加します。

**グローバル保持:**

- チェックを入れると、この変数に対するすべての変更が保存され、スクリプト終了後や電源を切って再起動しても変更後の値が保持されます。
- チェックを入れない場合、この変数の変更はスクリプト実行中のみ有効であり、スクリプトを終了すると初期設定値に戻ります。また、スクリプト実行中にグローバル変数の監視リストでこの変数のリアルタイム値を見ることはできず、プログラム変数を介してのみ確認できます。

## 使用制限:

- ローバル変数は中国語と英語のみサポートされています。
- **table型のグローバル変数**を使用する際、以下のような状況を避けてください。これらが発生すると、ロボットがエラーを報告してプロジェクトが停止します。以下では、`table_1`および`table_2`は追加済みのtable型グローバル変数を指します。
  - **変数値のネスト**

```
-- 正しい使用方法：テーブルの構成が定数または他の変数に割り当てます。
table_1[1] = {1,2,3}
table_1[2] = table_2
-- 間違った使用方法：テーブルの構成がテーブル自体に割り当てます。
```

```
table_1[1] = table_1 -- table_1の構成をtable_1自体に割り当てるとエラーが発生します。
```

# 8.7 プログラム変数

この画面は、実行中のプロジェクトで監視する必要がある変数を設定するためのものです。変数名を指定して、対象の変数のタイプや値を監視し、監視する変数名を変更することも可能です。


**追加**ボタンをクリックすると、プログラム変数を新規追加できます。変数を選択した後、**編集**ボタンをクリックすると、変数の属性を変更できます。**削除**ボタンをクリックすると、選択した変数を削除できます。


変数名にはオブジェクトのキー値を入力することができます:

- 例えば、P1ポイントのジョイント座標を監視したい場合、変数に `P1.joint` を追加します。
- 例えば、P1ポイントのX軸座標を監視したい場合、変数に `P1.pose[1]` を追加します。

追加が完了すると、変数リストに追加済みの変数が表示されます。

## プログラム変数の監視

プログラム変数の監視はスクリプト実行中に開始され、1秒ごとにポーリングを行います。スクリプトが一時停止または停止した場合、ポーリングも停止します。 追加されたプログラム変数に値が存在しない場合、実行中のプログラムではタイプと値が「-」と表示されます。


## プログラム変数のタイプ

プログラム変数は、出所に基づいて以下の3種類に分類されます：`local`、`upvalue`、`global`。同じ名前の変数が存在する場合、優先順位は前者が最も高いため、表示および変更されるのは現在の最も高い優先順位の変数となります。

監視および表示が可能なLuaのネイティブ変数タイプは以下の通りです：

- **nil**：空（有効な値がないこと）を表します。例えば、値が代入されていない変数を出力すると、`nil`が表示されます。
- **boolean**：ブール値で、`true`（真）と`false`（偽）の2つの値を取ります。Luaでは、`false`と`nil`が偽と見なされ、それ以外はすべて真と見なされます（`0`も真と見なされます）。
- **number**：数値で、倍精度の浮動小数点数を使用し、さまざまな演算をサポートします。
- **string**：文字列で、数字、アルファベット、アンダースコアから成る文字の並びです。
- **table**：テーブル（配列を含む）。`table`タイプは内容を展開して確認でき、子要素の変更も可能です。

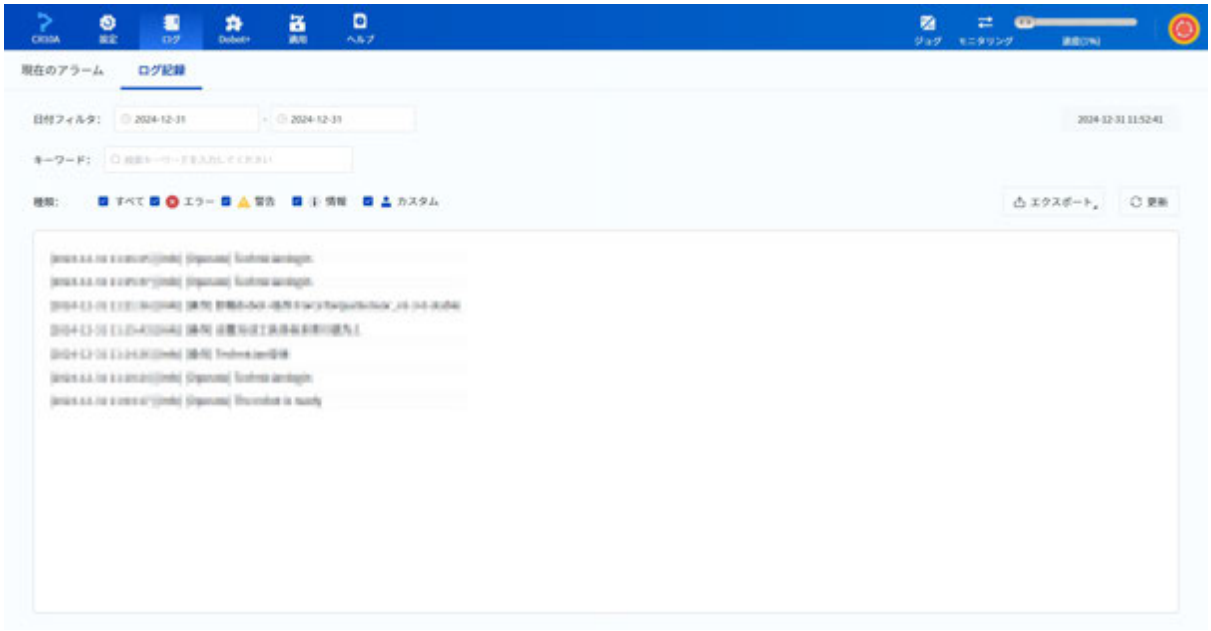
## 使用制限

- スクリプトが実行中、一時停止中、または停止中のいずれの場合でも、プログラム変数の追加、削除、変更が可能です。
- プログラム変数は最大20個まで追加可能で、変数名の長さは最大256バイトまでに制限され

ています。

- プログラム変数は中国語と英語のみ対応しています。

# 9 ログ




ログ記録画面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、ログタイプで検索することができます。

ログタイプの意味は以下の通りです:

- エラーログ: ソフトウェアがエラー動作を行った場合や、ロボットがアラーム状態になった際の関連アラーム情報。
- 警告ログ: ソフトウェアが異常動作を行った場合や、ロボットが異常状態になった際の関連警告情報。ただし、異常状態は後続の動作実行には影響しません。
- 操作ログ: ロボットの状態が変化した際に記録される情報。
- カスタムログ: ユーザーが `Log(value)` コマンドを使用して出力したログ情報。

## ログのエクスポート

**USBにエクスポート**: フィルタ条件に一致するログをコントローラーのUSBポートに接続されたストレージデバイスにエクスポートします。

**ローカルにエクスポート**: フィルタ条件に一致するログをPCのローカルストレージにエクスポートします。

**メーカーログをUSBにエクスポート**: コントローラー内のすべてのログをUSBにエクスポートします。


**説明:**

- 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含む場合、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、起動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が隠しパーティションである場合、Windows上でエクスポートされたログを直接確認できないことがあります。
- メーカー用ログのエクスポートが成功すると、USBメモリのルートディレクトリに「logs」という名前のフォルダが作成され、そこに保存されます。再度メーカー用ログをエクスポートすると、新しい「logs」フォルダが元のフォルダを上書きします。
- ログのエクスポート中にUSBメモリを抜かないようにしてください。そうしないと、USBメモリ内のディスクファイルが破損する可能性があります。

特定のシーン（例えば、ロボットアームが自動運転中に制御ソフトウェアがロボットアームに接続されている場合）では、ログが自動で更新されず、手動で **更新** をクリックして最新のログ記録を取得してください。

# 10 設定

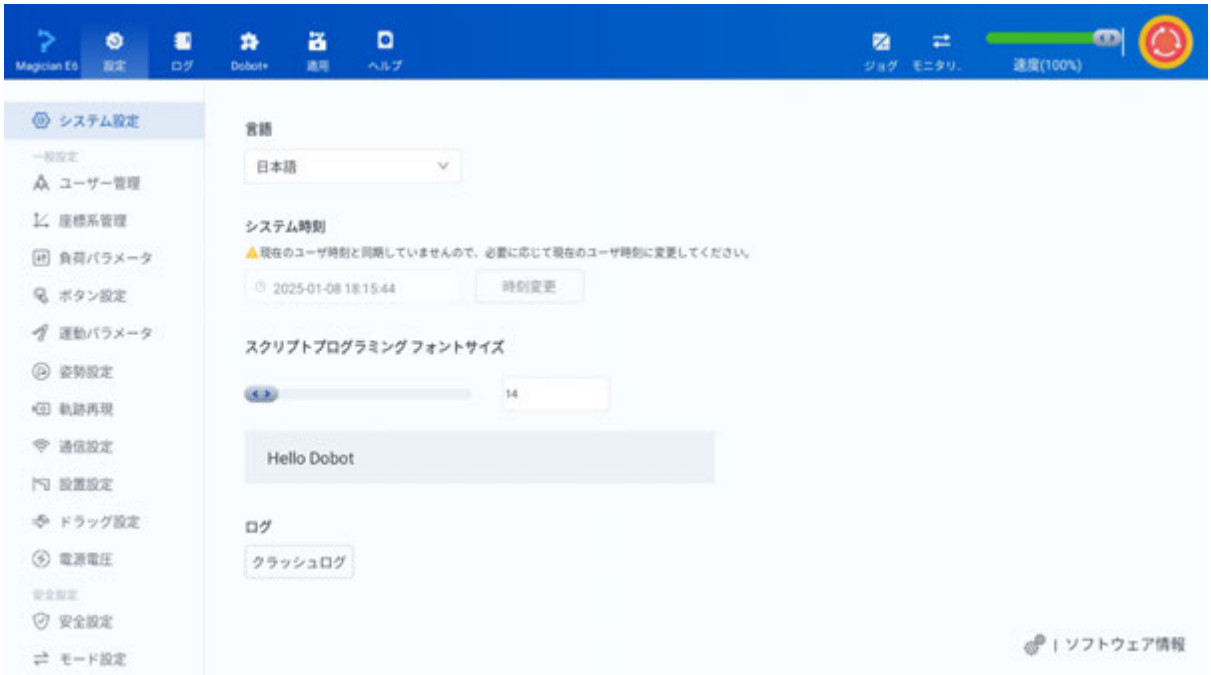

- 10.1 システム設定
- 10.2 ユーザー管理
- 10.3 座標系管理
- 10.4 負荷パラメータ
- 10.5 ボタン設定（Magician E6）
- 10.6 運動パラメータ
- 10.7 姿勢設定
- 10.8 軌跡再現
- 10.9 通信設定
- 10.10 設置設定
- 10.11 ドラッグ設定
- 10.12 電源電圧（直流コントローラー/Magician E6）
- 10.13 安全設定
- 10.14 モード設定
- 10.15 0 点校正
- 10.16 高度な機能
- 10.17 ファイル移行
- 10.18 ファームウェアアップグレード

# 10.1 システム設定

システム設定ページでは、ソフトウェアインターフェースの表示言語、システム時刻、スクリプトプログラミングのフォントサイズなどを変更できます。

**PC端末**


**モバイル端末**



**システム時間**：コントローラーの現在のシステム時間を表示します。デフォルトモードまたは手動モードではプロジェクトが実行されていない場合に変更可能です。現在のデバイスのシステム時間とコントローラーのシステム時間が一致していない場合、ページに警告が表示されます。一致させることを推奨します。

**スクリプトプログラムのフォントサイズ**：スクリプトプログラミング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、設定可能範囲は[12～50]です。

**ログ**：モバイル端末向け機能で、クラッシュログをアップロードします。インターネット接続が必要で、ロボットに接続していない状態で操作できます。

**次回ソフトウェア起動時に自動接続をデフォルトにする**：チェックを入れると、次回ソフトウェア起動時に現在接続されているロボットアームへの自動接続を試みます。PC版でのみ対応しています。

画面右下のアイコンとテキストはクリック可能で、機能は以下の通りです：

 ：メーカー機能を使用するにはメーカーのパスワードが必要です。技術サポートの指導のもとでご使用ください。

**バージョン情報**：DobotStudio Proの構成コンポーネントと免責事項を確認します。

以下の機能はPC版のみ対応しています。

**購読**：ソフトウェアの更新情報や製品ニュースを購読できます。

**フィードバック**：ソフトウェアの使用に関する問題を報告します。

**DOBOT公式サイト**：クリックするとブラウザが開き、DOBOT公式サイトにアクセスします。

# 10.2 ユーザー管理

ユーザーが管理者としてログインすると、この画面で機能の権限とパスワードを管理でき、カスタム機能の追加または削除を対応しています。

## デフォルト役割を設定する

画面左上にデフォルト機能、つまりロボットに接続するたびに自動的にログインする機能が設定することができます（管理者としての設定はできません）。デフォルト機能で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

＋ **役割の追加**をクリックし、最大2つのカスタム役割を追加することができます。

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 文字以内の英字と数字の組み合わせが対応しています。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は必ず有効になります。**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。変更する場合、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りません。**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後、もう一度**パスワードを変更する**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。古いパスワードを入力する必要がなく、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、下記の画面が表示されます。（カスタム役割の場合、追加した後に表示されます）。

すべての役割の権限は変更することができます。**デフォルト設定を復元する**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。

変更が完了した後、 **保存**をクリックすると、設定を有効になります。

## デフォルトの役割権限

デフォルトの役割権限は下表の通りです。**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 | 操作者 |
|---|---|---|---|---|
| 基本設定 | プロジェクトを開く/実行/停止 | √ | √ | √ |
| | イネーブル、アラームクリア、手動/自動モード切り替え、速度比率調整 | √ | √ | √ |
| | ロボットジョグ | √ | √ | √ |
| | ログの確認・エクスポート | √ | √ | √ |
| | オンライン/TCPモード切り替え、IO/Modbus設定のオン/オフ | √ | √ | X |
| | IO（コントローラーDI/DO、コントローラーAI/AO、末端I/O、安全I/O）設定 | √ | √ | X |
| | Modbus設定 | √ | √ | X |
| | グローバル変数の設定 | √ | √ | X |
| | DOBOT+プラグイン、末端ボタン設定 | √ | √ | 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 |
| | ボタン設定 | √ | √ | X |
| | 軌跡の記録と再現 | √ | √ | X |
| | 通信設定 | √ | √ | X |
| | 電源電圧 | √ | √ | X |
| 詳細設定 | パッケージ・0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 |
| | ドラッグ設定 | √ | X | X |
| | 安全設定 | √ | X | X |
| | 手動・自動モード設定 | √ | X | X |
| | 0点校正 | √ | 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として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 座標系管理


- 10.3.1 ユーザー座標系
- 10.3.2 ツール座標系

# 10.3.1 ユーザー座標系

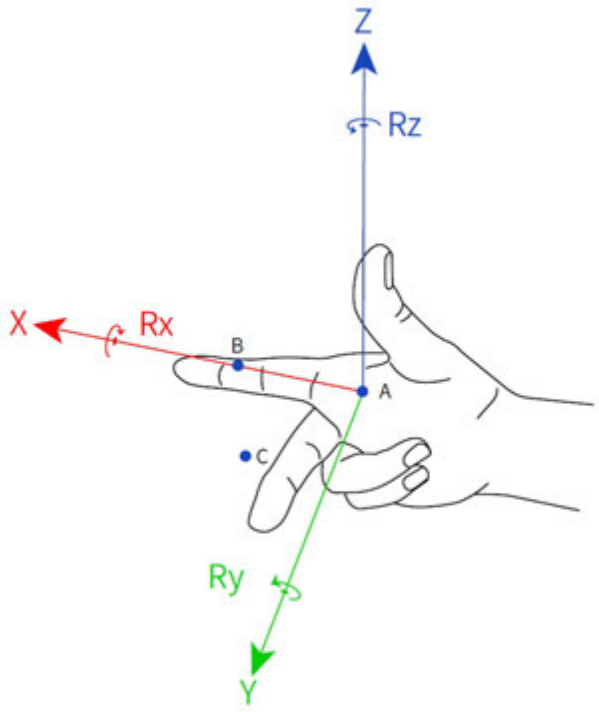
ワークの位置が変化した場合や、ロボットアームの運転プログラムを複数の同タイプの加工システムで繰り返し使用する必要がある場合、ユーザー座標系を設定する必要があります。これにより、すべてのパスがユーザー座標系に従って同期的に更新され、ティーチングプログラミングが大幅に簡素化されます。

現在のシステムでは、最大51個のユーザー座標系（0～50）をサポートしています。ユーザー座標系**0**は基準座標系（各ロボットのハードウェアマニュアルを参照）であり、変更することはできません。

ユーザー座標系の原点と各軸の方向は、同じロボット姿勢であっても、異なるユーザー座標系では異なる座標値を持つようにカスタマイズすることができます。ユーザー座標系の値は、ユーザー座標系0を基準としたユーザー座標系のオフセットと回転角度を表します。

> ⚠ **注意:**
> ユーザー座標系を設定する際、ポイント取する際の基準座標系がユーザー座標系**0**であることを確認してください。

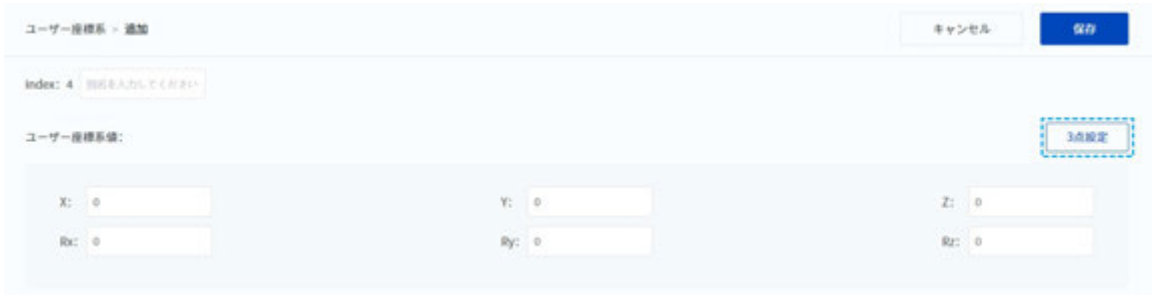
ユーザー座標系は、3ポイントティーチング法を使用して生成することを推奨します：ロボットを**A**、**B**、**C**の任意の3ポイントに移動させます。このうち、**A** 点が原点として使用され、**AB** の2点間の接続線がユーザー座標系のX軸の正の方向として決定し、**C**点がX軸に沿って垂直線を引き、Y 軸の正の方向として決定し、右手の法則に従ってZ軸の方向を決定します。




## ユーザー座標系の作成

1. 座標系画面で**追加**をクリックします。下図の通りです。

2. ユーザー座標系追加画面で座標系値の右上の**3点設定**をクリックします。



**説明:**

ユーザーはX、Y、Z、Rx、Ry、Rzの値を直接編集し、**保存** をクリックすることもできます。X、Y、Zはユーザー座標系の原点が基準座標系内での位置を表し、Rx、Ry、Rzはユーザー座標系が基準座標系に対してX->Y->Zの順で回転する角度をそれぞれ表します。

3. イメージ図に示すように、ロボットを対応するポイントに移動させた後、**ポイント取得**をクリックします


> **説明:**
> **移動先** を長押しすると、ロボットアームを取得済みの位置に移動させることができます。

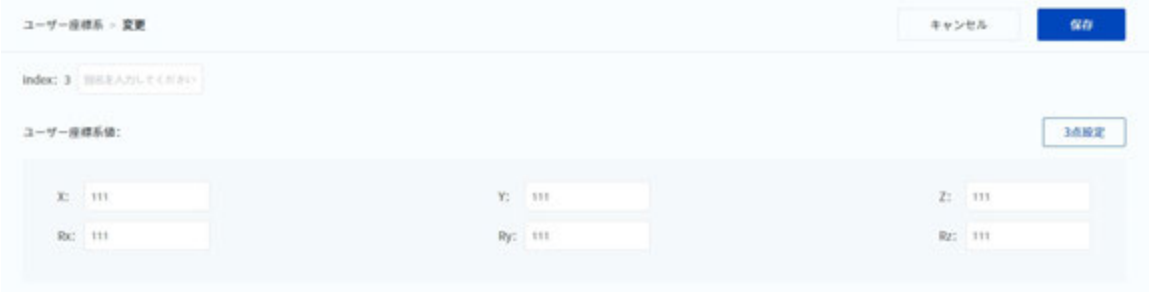
4. **確定** をクリックすると、ユーザー座標系の追加ページに戻り、座標系の値がキャリブレーションで生成された値に更新されます。この値を確認または編集することができ、3ポイント設定で生成された内容を確認するか、手動で座標系の値を変更することもできます。ユーザー座標系の変更手順については以下を参照してください。

5. **保存** をクリックすると、ユーザー座標系リストにこの座標系が追加されます。

## ユーザー座標系の変更

1. ユーザー座標系画面で座標系を選択した後、**変更**をクリックします。下図の通りです。


2. 選択したユーザー座標系の値が手動入力後に保存されたものである場合、編集ページのUIは追加ページと同じになります。この場合、値を直接編集するか、**3点設定** をクリックしてキャリブレーションを行うことができます。一方、選択したユーザー座標系の値が3ポイント設定によって生成されたものである場合、編集ページのUIは以下の図のようになります。




3. 座標系の右下の**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ポイント設定を表示し、変更が必要なポイント位置について再取得することができます。
4. **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。

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

5. **保存**をクリックすると、この座標系をツール座標系一覧に追加します。

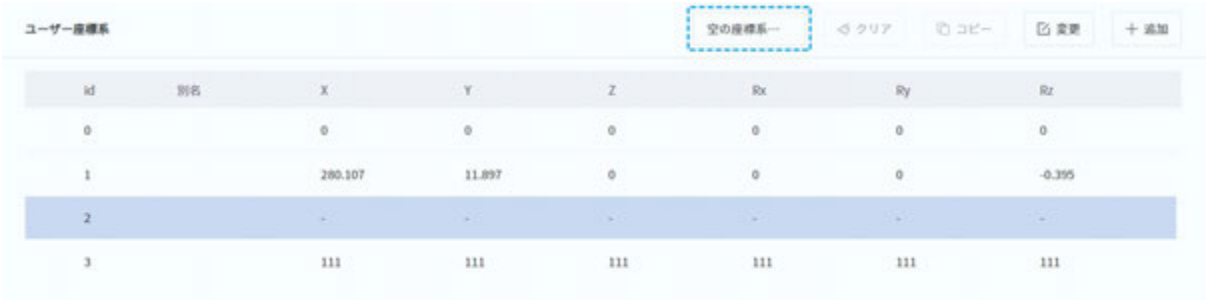
## ユーザー座標系のコピー

ユーザー座標系画面で座標系を選択してから**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コピー作成することができます。


## ユーザー座標系のクリア

ユーザー座標系ページで座標系を選択した後、**クリア** をクリックして確認すると、選択した座標系をクリアできます。クリアされた座標系はそのIDを引き続き占有しますが、座標系データのみが消去されます（例：以下の図の座標系2）。この座標系が呼び出された場合、エラーが発生します。


**空の座標系を隠す**をクリックすると、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。当該ボタンが**空の座標系を表示**に変わり、クリックすると、空の座標系を再表示することができます。




空の座標系は変更可能、変更後に新しいデータに割り当てされます。

# 10.3.2 ツール座標系

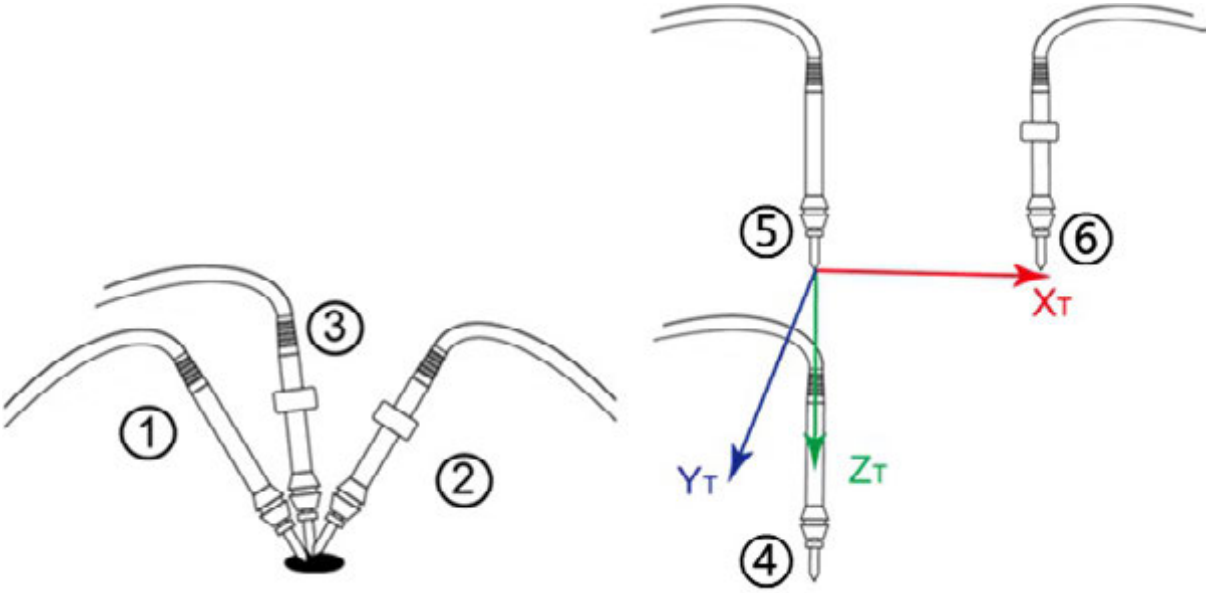
ロボットの末端にツール（溶接トーチ、ノズル、治具など）を取り付ける場合、プログラミングやロボットの動作のためにツール座標系を設定する必要があります。たとえば、複数の治具を使用して複数のワークを同時に処理する場合、各治具を独立したツール座標系に設定して処理効率を向上させることができます。

現在のシステムでは、最大51個のツール座標系（0～50）をサポートしています。ツール座標系**0**はフランジ座標系（各ロボットのハードウェアマニュアルを参照）であり、ツールを使用しない場合の座標系を指します。この座標系は変更できません。

ツール座標系は、TCP（Tool Center Point、通常はツールの作用点、例：吸盤の中心や溶接ガンの先端）を原点として設定される座標系で、ツールの位置と姿勢を表します。ツール座標系の値は、ツール座標系0に対するオフセット量と回転角度を示します。

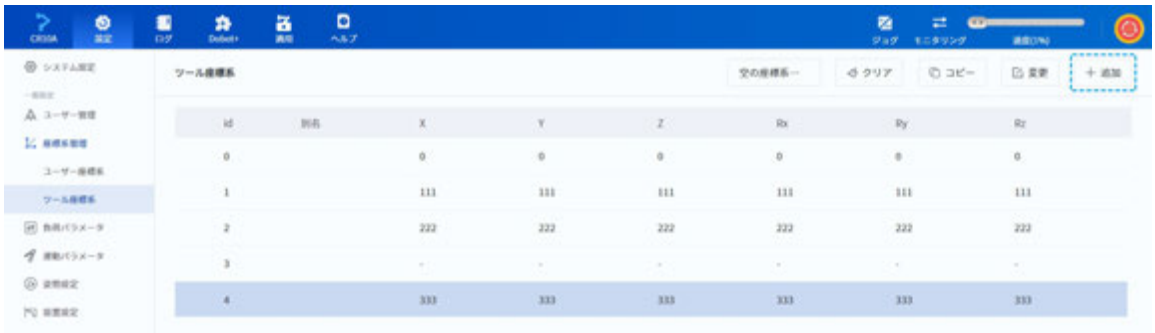
> ⚠ **注意:**
> ツール座標系を設定する際は、ポイントを取得する際の基準座標系がツール座標系**0**であることを確認してください。。

6軸ツール座標系は、6ポイントティーチング法「TCP+ZX」を使用して生成することを推奨します。ロボット先端にツールを取り付けた後、TCP が一致するようにツールの姿勢を調整します。空間内の同じポイントを異なる3方向（①②③）の基準点）に移動させ、工具位置オフセットを取得します。次に、他の3ポイント（④⑤⑥）に基づいてツール姿勢オフセットを取得します。ここで、④は①②③と同じポイントにします。

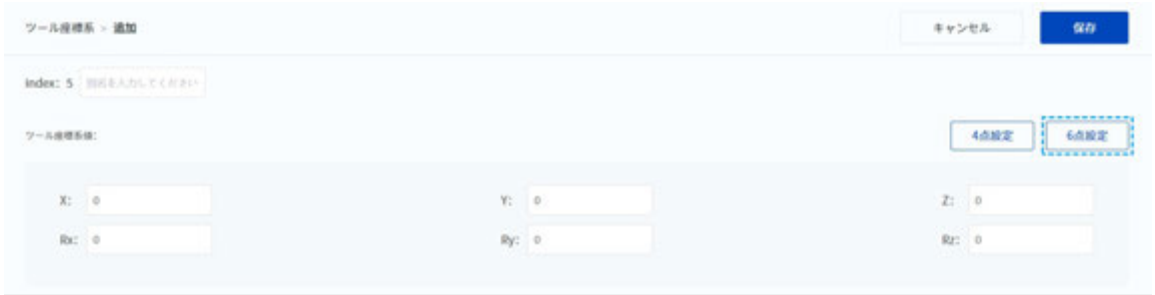


## ツール座標系の作成

1. ツール座標系画面で＋**追加**をクリックします。下図の通りです。



2. ツール座標系追加画面で座標系値の右上の**6ポイント設定**をクリックします。



> **説明:**
> 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**と操作方法が同じ、本文章ではその説明は省略します。

3. イメージ図に示すように、ロボットを対応するポイントに移動させた後、**ポイント取得**をクリックします。

> **説明:**
> **移動**を長押しすると、ロボットを取得したポイント位置に移動させることができま

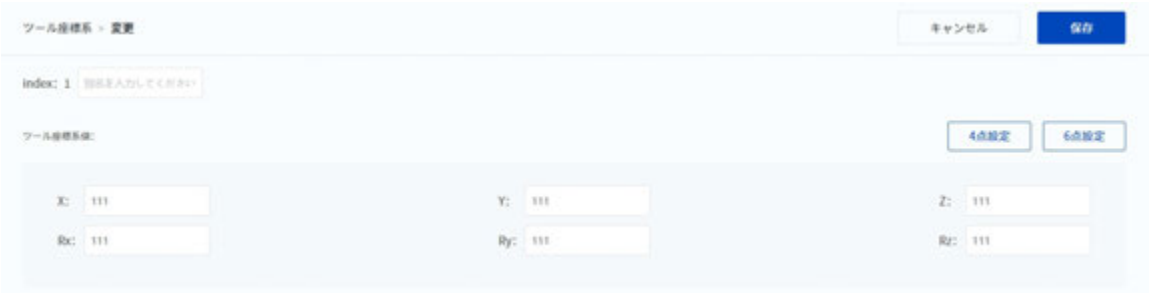
す。

4. 該座標系を生成した6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. **保存**をクリックすると、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. ツール座標系画面で座標系を選択した後、**変更**をクリックします。下図の通りです。

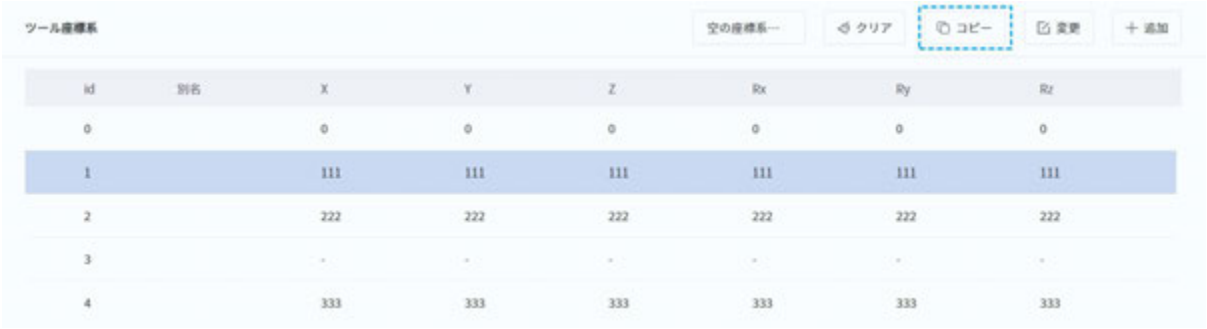

2. 選択したツール座標系の座標系値が手動入力後に保存された値である場合、編集ページの外観は追加ページと同じになります。この場合、直接編集するか、**4ポイント設定/6ポイント設定**をクリックしてキャリブレーションを行うことができます。 一方、選択したツール座標系の座標系値が4点/6点設定で生成された場合、編集ページの外観は以下のようになります。**6ポイント設定を確認**という文字は、生成方法に応じて変更されます。




3. **6ポイント設定を表示**をクリックすると、該座標系値を生成した6ポイント設定を表示し、変更が必要なポイント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. **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ポイント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. **保存**をクリックすると、この座標系をツール座標系一覧に追加します。

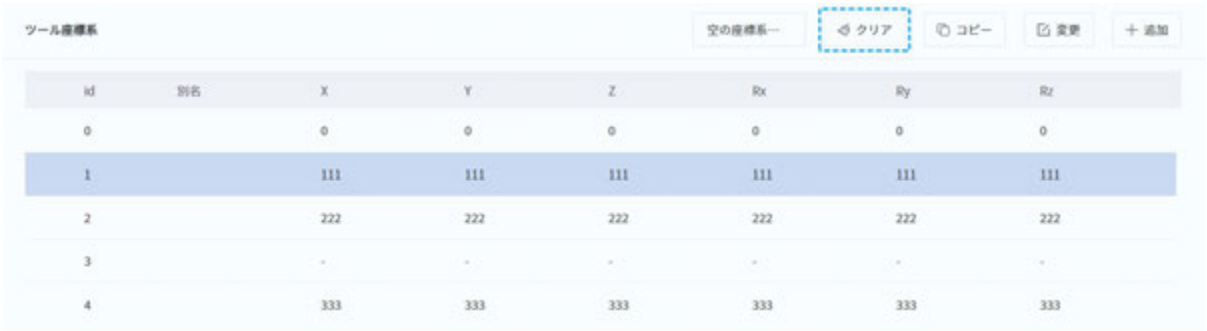
## ツール座標系のコピー

ツール座標系画面で座標系を選択してから**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コピー作成することができます。

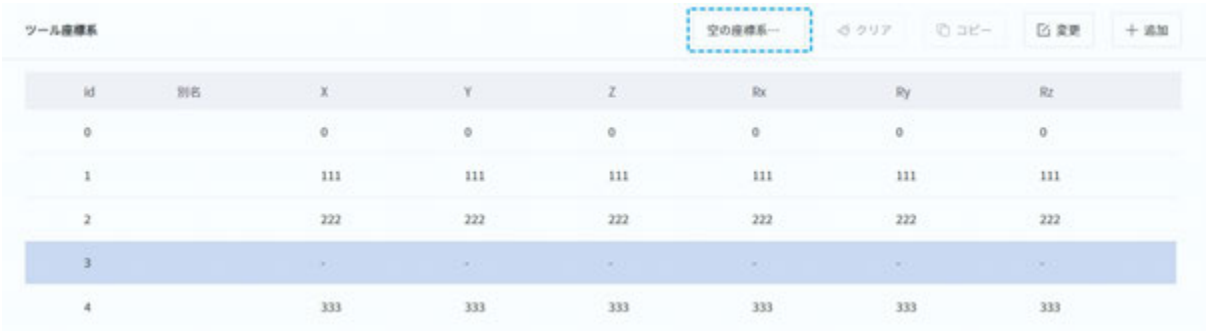



## ツール座標系のクリア

ツール座標系ページで座標系を選択した後、 **クリア**をクリックして確認すると、選択した座標系をクリアできます。クリアされた座標系は、そのIDを引き続き占有しますが、座標系データのみが消去されます（例：以下の図の座標系3）。この座標系が呼び出された場合、エラーが発生します。




**空の座標系を隠す**をクリックすると、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。当該ボタンが**空の座標系を表示**に変わり、クリックすると、空の座標系を再表示することができます。




空の座標系は変更可能、変更後に新しいデータに割り当てされます。

# 10.4 負荷パラメータ

負荷パラメータは、ロボットの末端の負荷（治具を含む）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。

> **説明:**
> 本画面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、ロボットをイネーブルにするときに表示される負荷設定画面で対応するパラメータグループを選択してください。

> **注意:**
> 荷重設定を誤るとロボットの性能が低下し、異常な衝突検出アラームが発生したり、ドラッグ時に機体が制御不能になる可能性があります。

単击 **新增**可以新增一组负载参数；单击选中一组参数组后，单击 **修改**可以修改选中的参数组，单击 **删除**可以删除选中的参数组。

## パラメータグループの追加/変更


パラメータグループの別名は変更可能で、有効化やプログラミング時にパラメータグループを参照する際に使用されます。

負荷パラメータの設定には、負荷識別と手動変更の2つの方法があります。

**負荷識別**

ロボットがイネーブルであり、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を自動的に認識することができます。

> **説明:**
> - 負荷識別を行う前に、ロボットの設置角度が正しく設定されていることを確認してください。
> - Magician E6は負荷識別機能をサポートしていません。

1. **負荷識別**をクリックすると、自動識別ウィンドウを開きます。
2. システムは7つのデフォルトポイントを自動的に生成します。ポイントを選択すると、その姿勢図が表示されます。**移動**を長押しすると、ロボットをそのポイント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ポイントに移動できない場合、そのポイント図を参照して、条件を満たす別のポイントにロボットをジョグさせることができます。その後、デフォルトのポイントに**上書き**します。**デフォルトポイントに復元**をクリックすると、上書きされたポイントをデフォルトのポイントに戻すことができます。
4. **ワンクリック自動識別**をクリックして確認すれば、ロボットが各ポイントまで順次移動し、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、自動識別画面に残ります。各ポイント図を参照して、そのポイント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ポイントを変更してからもう一度**ワンクリック自動識別**を実行してください。


**手動変更**

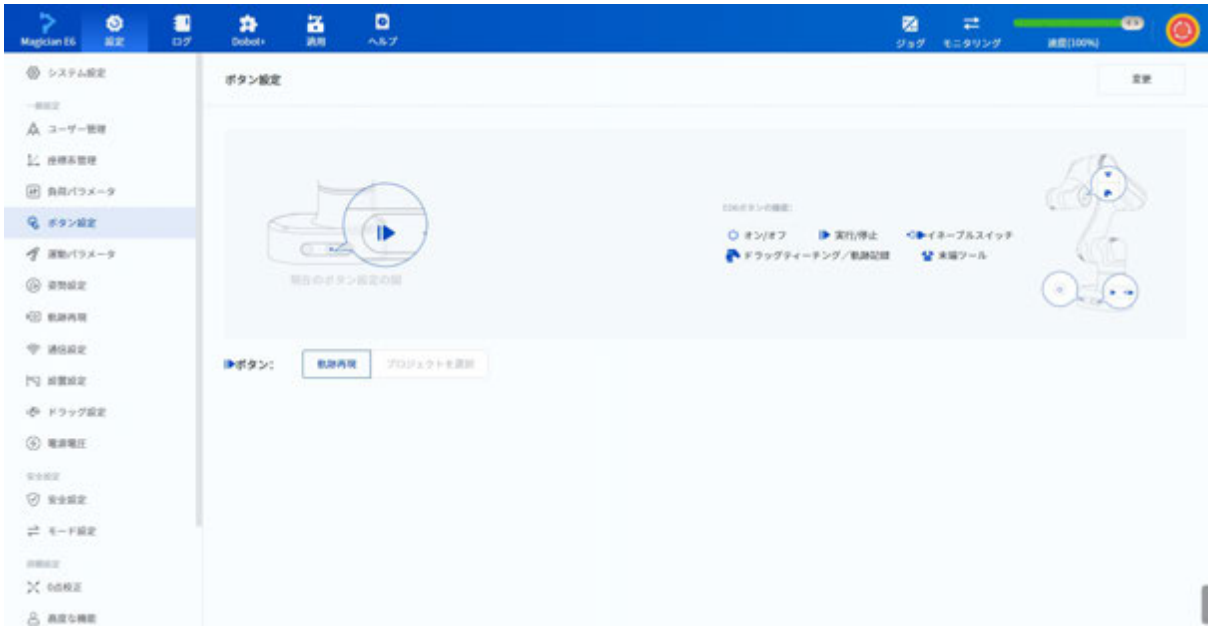
以下の負荷パラメータを手動で入力します:

- **X/Y/Z方向の中心オフセット距離**：末端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。
- **ペイロード**：末端治具の重量とワークの重量の合計です。ロボットの最大許容負荷を超えることはできません。

設定変更後、**保存**ボタンをクリックして設定を完了してください。

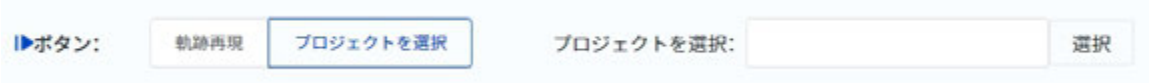
# 10.15 ボタン設定（Magician E6）

ロボット機種がMagician E6の場合、ベースにある運転/停止ボタンの機能を設定することができます。




右上の**変更**ボタンを押すと、運転/停止ボタンの機能を変更するとこができます。

- デフォルトの機能は軌跡再現です。最近の一回の録画された軌跡を再現します。
- 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、ロボットの軌跡再現中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。

# 10.6 運動パラメータ

ロボットは出荷時に、通常の作業条件に最適な動作パラメータが設定されています。特別な要件がない限り、これらの設定を変更することは推奨されません。ユーザーがロボットの速度が高すぎると感じる場合は、実際のニーズに応じて速度を下げることができます。速度を上げる必要がある場合は、技術サポートに連絡し、加速プランについて相談してください。

CRAシリーズとMagician E6では、ページの内容が異なります。

# 10.6.1 運動パラメータ（CRA）

## 再現パラメータ

再現パラメータは、ロボットが**プロジェクトを実行**したり、**TCP_IP動作コマンド**を使用する際の動作パラメータです。


### トルク制限

この機能はロボットの安定した動作を保証するために非常に重要であり、有効化することを推奨します。有効化すると、ロボットのアルゴリズムが実際の運転状況に応じて加速度と加加速度を調整し、トルクオーバーに起因するアラームを回避します。

### 再現速度

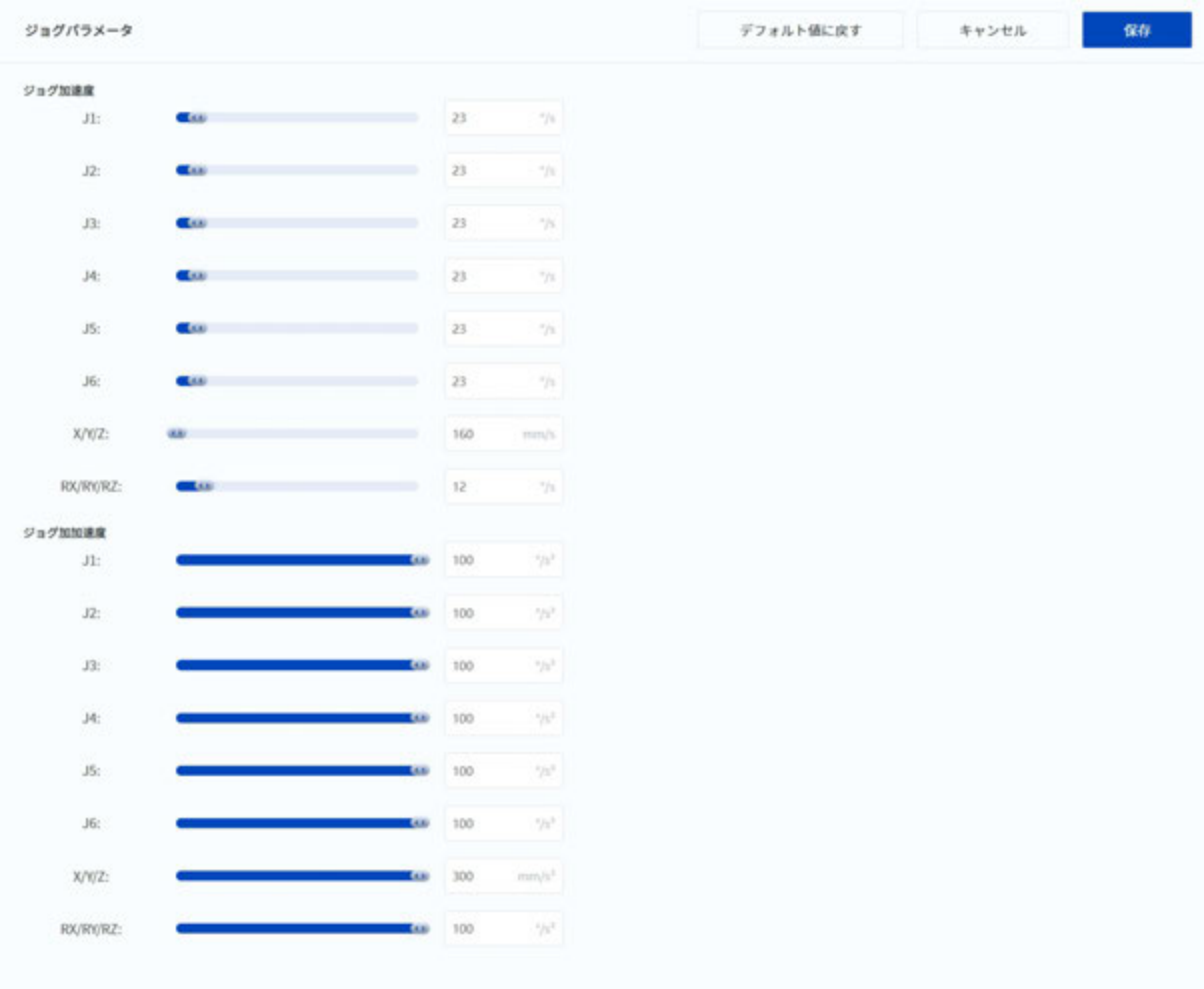
再現動作時のロボット各関節の最大速度を設定します。通常モードと縮減モードの最大速度をそれぞれ設定する必要があります。縮減モードの最大速度は、通常モードの最大速度を超えてはいけません。

このページでは関節速度のみを設定できます。デカルト座標系での最大TCP速度は、**安全制限**ページで設定してください。

**再現加速度/再現加加速度**

再現動作時の加速度および加加速度の最大値を設定します。関節運動とデカルト運動の値をそれぞれ設定する必要があります。

## ジョグパラメータ




**ジョグパラメータ**は、ロボットが**ジョグ/インクリメンタル移動**および**移動先**の動作を行う際の動作パラメータです。

**ジョグ速度**

ジョグ動作の最大速度を設定します。関節速度とデカルト速度をそれぞれ設定する必要があります。

**ジョグ加速度**

ジョグ動作の最大加速度を設定します。関節加速度とデカルト加速度をそれぞれ設定する必要があります。

## サイクルタイムの影響要因

- 実際の操作では、ロボットの最大動作速度は関節の最大速度とTCPの最大速度の両方によって決まります。関節の最大速度は**動作パラメータ**画面で設定可能で、TCPの最大速度は**安全制限**（TCP速度、モーメント、停止時間、停止距離など）の影響を受けます。
- ロボットの最大加速度および加加速度は、ジョグパラメータだけでなく、**トルク制限**機能が有効かどうかにも依存します。この機能は、操作時のロボットの安全性と安定性を確保するために役立ちます。

# 10.6.2 動作パラメータ（Magician E6）


**ティーチング設定**は、ロボットが**ジョグ/インクリメンタル移動**および**移動先**の動作を行う際の動作パラメータです。関節またはデカルト座標系における最大速度、最大加速度を設定することができます。

**再現設定**は、ロボットが**プロジェクトを実行**したり、**TCP_IP動作コマンド**を使用する際の動作パラメータです。関節またはデカルト座標系における最大速度、最大加速度、最大加加速度を設定することができます。

設定の変更が完了したら、**保存**をクリックして変更内容を保存します。**キャンセル**をクリックすると、今回の変更を取り消します。**デフォルト値に戻す**をクリックすると、パラメータを出荷時のデフォルト値にリセットすることができます。

# 10.7 姿勢設定

このページは、ロボットをさまざまな出荷時にプリセットされた姿勢に移動させるために使用されます。

- **パッキング姿勢** は、ロボットの占有スペースを縮小し、梱包や輸送を容易にします。
- **0点姿勢** は、各関節の角度がすべて0度になります。

を長押しすると、ロボットを対応する姿勢に移動させることができます。

> **説明:**
> ジョグ操作画面を使用して、ロボットをこの姿勢に移動させることもできます。

# 10.8 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されます。

右上の ＋ **新しい軌跡を作成** をクリックすると、ロボットアームがドラッグモードに入り、ユーザーはこの状態でロボットアームを手動でドラッグできます。ドラッグした軌跡は記録されます。軌跡の記録中は、50msごとに1点を記録し、1つの軌跡につき最大10,000点（約500秒）まで記録可能です。

> ⚠ **注意:**
> 軌跡の記録中にユーザー座標系やツール座標系を切り替えないでください。記録された軌跡が再現できなくなる可能性があります。

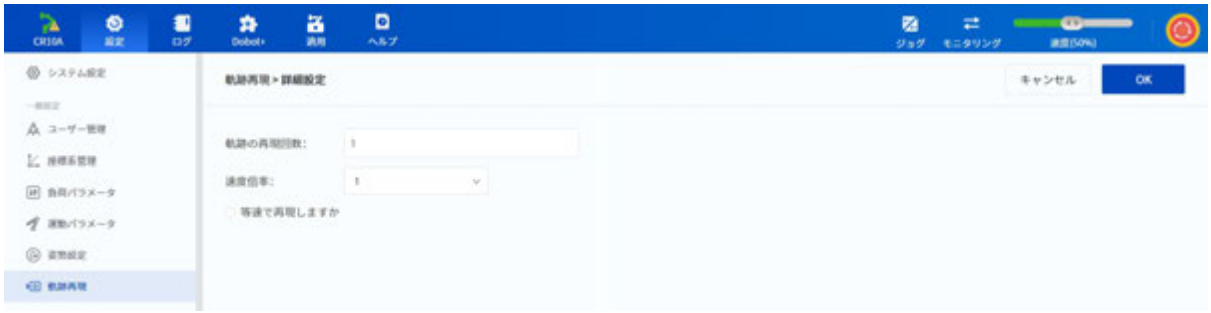
記録したい軌跡が完了したら、**保存** をクリックすると、ロボットアームがドラッグモードを終了し、軌跡ファイルリストに新しい記録が追加されます。

- 軌跡ファイル名の横にある をクリックすると、軌跡ファイル名を変更できます。
- **軌跡再現** をクリックすると、ロボットアームが記録された軌跡を再現します。再現中はいつでも停止できます。
- 軌跡ファイルを選択して **削除** をクリックすると、その軌跡を削除できます。
- **更新** をクリックすると、コントローラーの最新の軌跡ファイルリストを取得できます。

> **補足:**
> 保存された軌跡ファイルは、グラフィックプログラミングの軌跡再現指令、またはスクリプトプログラミングの軌跡再現指令で呼び出すこともできます。

**高度な設定** をクリックすると、軌跡再現の方法に関する設定を行えます。この設定は、DobotStudio Proのインターフェースやロボットアームのエンドボタンを使って軌跡再現を開始する場合に有効です。

- **軌跡再現回数** は、**軌跡再現** をクリックした後にロボットが軌跡を再現する回数を示します。回数が1を超える場合、ロボットアームは1回の軌跡再現が終わると、関節動作で軌跡の始点に戻り、次の再現を開始します。
- **速度倍率** は、**等速再現** を選択しない場合にのみ有効です。ロボットアームは元の速度を比例縮小（全体の速度設定には影響されません）して軌跡を再現します。
- **等速再現** を選択すると、ロボットアームは全体の速度設定に基づいて等速で軌跡を再現します。




> **補足:**
> ユーザーはロボットのエンドボタンを使用し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参照してください）。DobotStudio Proインターフェースとロボットエンドボタンを使用した場合の軌跡記録および再現の違いは以下の通りです:
> - DobotStudio Proインターフェースで生成される軌跡ファイル名は保存時の「年-月-日-時-分-秒」の形式で、保存するたびに新しいファイルが作成されます。一方、ロボットエンドボタンで生成される軌跡ファイル名はTrackRecord.csvで、保存するたびに前回のファイルが上書きされます。
> - DobotStudio Proインターフェースでは再現する軌跡を選択できますが、ロボットエ

ンドボタンではTrackRecord.csvの軌跡のみを再現できます。

# 10.9 通信設定

**通信設定**は、現在接続されているコントローラーの通信パラメータに関連するインターフェースを設定するために使用されます。例えば、コントローラーのLAN1のIPアドレス、バスモード、およびWiFi関連の属性を設定します。


## 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することができ、TCP、UDP、またはModbusプロトコルをサポートします。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、およびゲートウェイを変更することができます。外部デバイスに接続する際、IPアドレスは外部デバイスのIPアドレスと同じネットワークセグメント内にあり、かつ重複していない必要があります。

- ロボットと外部デバイスが直接接続されている場合、またはスイッチを介して接続されている場合は、**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接続する場合に変更が必要）、およびデフォルトゲートウェイを設定し、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します。
- ロボットと外部デバイスがルーターを介して接続されている場合は、**自動取得**（ルーターによるIPアドレスの自動割り当て）を選択します。

## バス通信

バス通信機能を設定します。**オフ**、**Profinet**、または**EtherNet/IP**に設定することができます。

**Profinet**または**EtherNet/IP**に設定する場合、バス通信が切断された際のロボットの動作を設定する必要があります:

- **動作を継続**: バス通信が切断されても何も処理せず、プロジェクトの実行を続行します。
- **一時停止**: バス通信が切断された場合、実行中のプロジェクトを一時停止します。
- **停止**: バス通信が切断された場合、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）」を参照してください。

> ⚠**注意:**
> バス通信をProfinetに設定する場合、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、コントローラー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することができます。ユーザーはWiFiを有効または無効にし、WiFiの名称やパスワードを変更することができます。 をクリックすると、現在のパスワードを表示できます。

Magician E6のWiFiモジュールは、ユーザーが別途購入して取り付ける必要があり、このページでWiFiのオン/オフを切り替えることはできません。

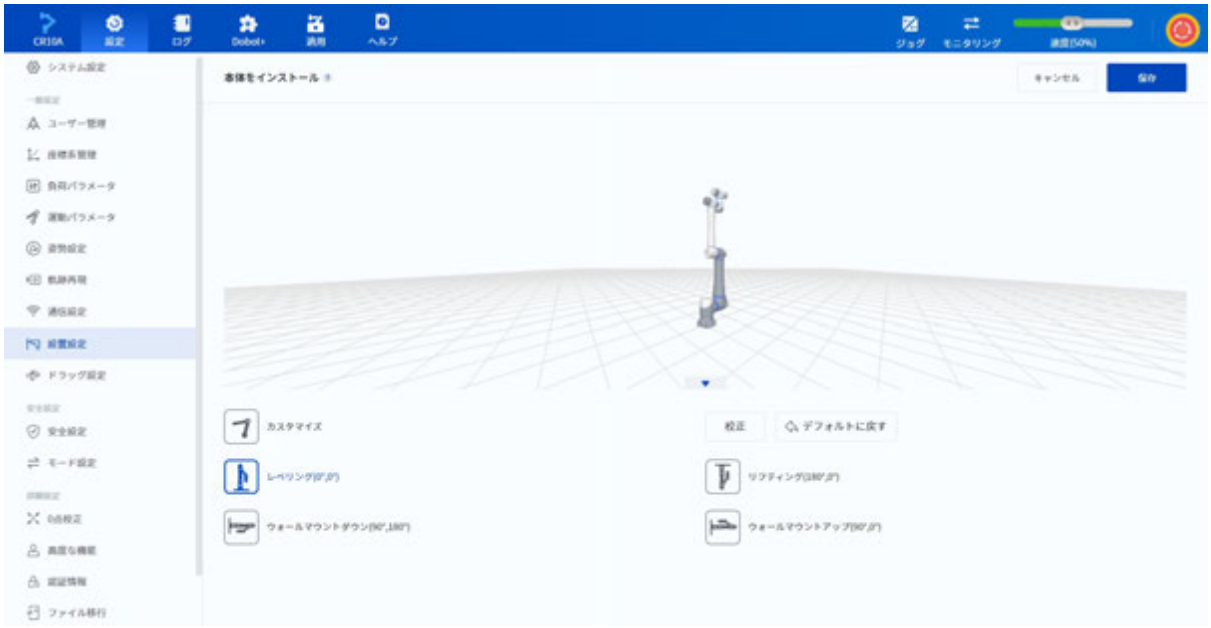
> ⚠ **注意:**
> WiFi設定を変更すると、ソフトウェアとロボットの接続が切断される可能性があります。5秒後に再接続を試みてください。

# 10.10 設置設定

通常、ロボットは安定したテーブルまたは地面に設置されます。この場合、この画面での操作は必要ありません。ロボットを天井や壁、または一定の角度で設置する場合は、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。

設置ル設定の主な機能は、ロボットに正しい重力方向を学習させ、ソフトウェア内の3Dモデルを正しい角度で表示できるようにすることです。

> ⚠ **注意:**
> 設置設定を誤ると、異常な衝突検出アラームが発生したり、ドラッグ時に本体が暴走したりする可能性があります。



ユーザーは手動または自動の2つ方法で校正を行うことができます。

## 手動校正

実際の設置姿勢により、画面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。各姿勢の詳細な説明は、画面タイトルの右側の ? をクリックして確認することができます。

- **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。

## 自動校正

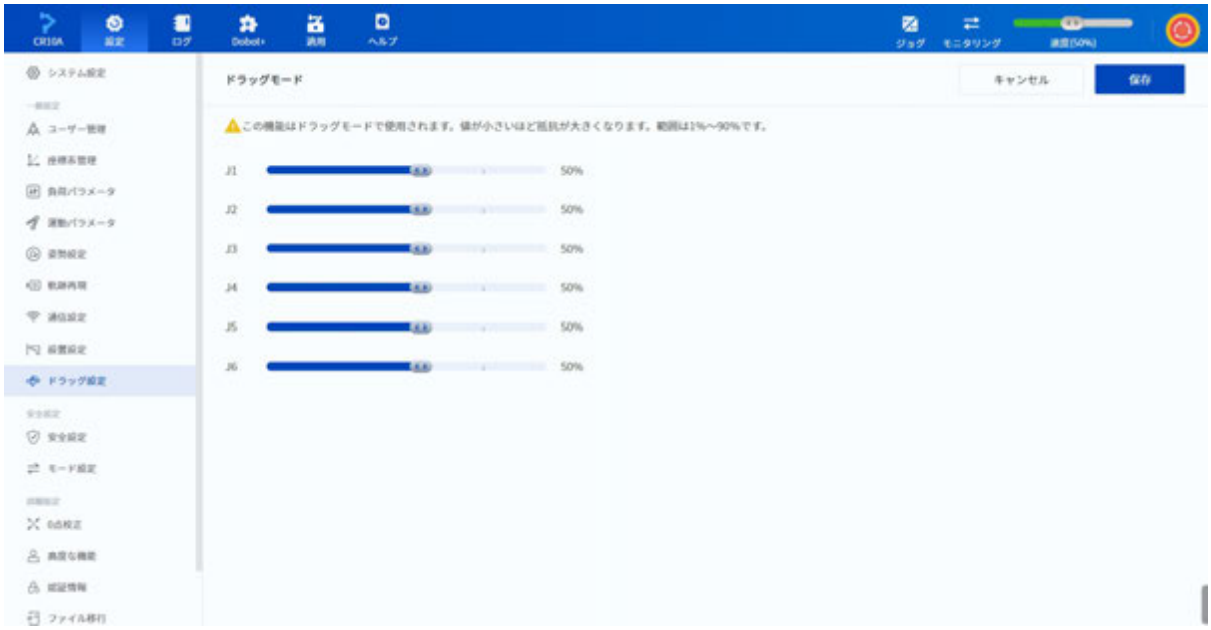
ロボットを設置してイネーブルにした後、**校正**をクリックし、ポップアップダイアログボックスに従って操作し、傾斜角度と回転角度を取得します。


**デフォルトに戻す**をクリックすると、校正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、ロボットの末端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、技術サポートにご連絡ください。

# 10.11 ドラッグ設定

ドラッグ設定は、ロボットの各関節におけるドラッグ操作の感度を調整するために使用されます。感度が低いほど、ドラッグ操作中の抵抗が大きくなります。設定可能な範囲は1%～90%です。




**説明:**

- 感度設定は、関節のドラッグ操作が制限速度に達していない場合にのみ有効です。ドラッグ速度が制限速度に達すると、逆方向の抵抗が発生し、速度超過を防ぎます。
- 異なるロボットアームモデルごとに制限速度は異なります。J1～J3の制限速度は約30～40°/s、J4～J6の制限速度は約90～100°/sです。

# 10.12 電源電圧（直流コントローラー/Magician E6）

ロボットが直流コントローラーまたはMagician E6に接続されている場合、DobotStudio Proソフトウェアで電源電圧を設定する必要があります。この電圧範囲は、減速やブレーキ時にロボットが発生させる逆起電力を放出するためのフィードバック機能に関連しています。入力電源の実際の電圧範囲に基づいて設定を行い、過電圧保護による電源遮断やコントローラーの損傷を防いでください。

**設定** > **電源電圧** をクリックし、実際の入力電源の電圧範囲を入力します。

- CC262 DC版に接続する場合、設定可能な範囲は30～60Vであり、最小値≤最大値。
- Magician E6に接続する場合、設定可能な範囲は36～58Vであり、最小値≤最大値。

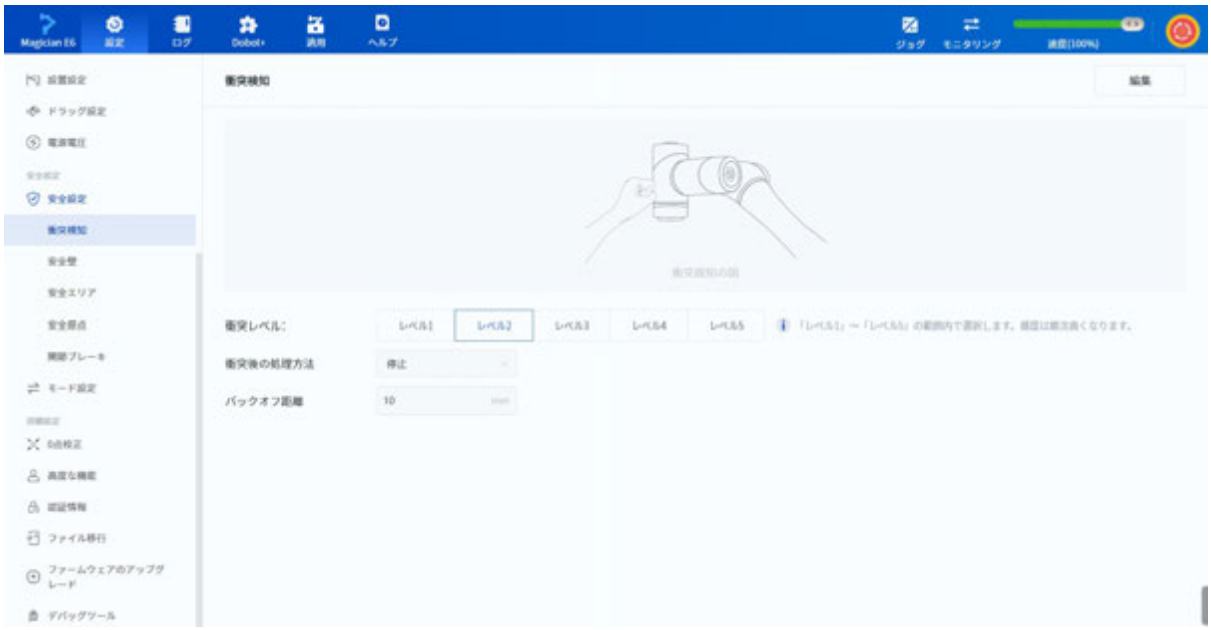
# 10.13 安全設定


- 10.13.1 衝突検出（Magician E6）
- 10.13.2 安全制限（CRA）
- 10.13.3 関節リミット（CRA）
- 10.13.4 安全壁
- 10.13.5 安全エリア
- 10.13.6 安全原点
- 10.13.7 関節ブレーキ

# 10.13.1 衝突検出（Magician E6）

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することができます。

> **説明:**
> 衝突検出を有効または無効にする必要がある場合は、技術サポートにお問い合わせください。



実際のニーズに応じて設定を行ってください。衝突レベルが高いほど、衝突検出をトリガーするのに必要な力は小さくなります。

- レベル5は、ロボットが低速で動作している場合のみに使用されます。この衝突レベルでは、ロボットが負荷をかけて高速/高加速度で動作しているときに、衝突検出が誤ってトリガされる可能性があります。
- ロボットが満載の場合、衝突レベルはレベル3以下に設定してください。

ジョグと自動運転時、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、衝突が検出された画面が表示される場合は、衝突の原因を取り除く必要があり、**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分後にリマインドしてください**をクリックしてください。


**自動実行中の衝突**

ロボットは**衝突後の処理方法**によって処理します:

- **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ます。この場合、実際の状況に応じて衝突の原因を取り除き、続行するか停止するかを選択してください。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ティーチング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。

# 10.13.2 安全制限（CRAシリーズ）

ロボットとユーザーの安全性を確保するために、ロボットシステムはロボットの動作時の関連パラメータを制限します。ユーザーは自分のニーズに応じて制限値を設定することができます。


**パラメータ設定**

- クイック設定とカスタム設定の2つの方法をサポートしています。
    - **クイック設定**：5つのプリセットレベルから選択します（以下では、最も緩い制限をレベル1、最も厳しい制限をレベル5と呼びます）。

> **説明:**
> - レベル5はロボットの低速運転時のみ使用してください。この衝突レベルでは、ロボットが負荷を伴う高速/高加速運転を行う場合に、誤って衝突検出が作動する可能性があります。
> - ロボットが最大積載量の場合は、衝突レベルをレベル3以下に設定してください。
> - プリセットレベルはあくまで推奨値であり、正確なリスク評価を代替するものではありません。

- **カスタム設定**：各制限パラメータの値を手動で入力します。クイック設定に切り替えると、パラメータは対応するプリセットレベルの値にリセットされます。

各制限パラメータの意味は以下の通りです（すべてのパラメータは通常モードと縮減モードそれぞれで制限値を設定する必要があります）：

- **TCP力**：ロボットのエンドツール中心点（TCP）に加わる最大力を制限します。
- **パワー**：ロボットの総消費電力の最大値を制限します。
- **TCP速度**：ロボット動作時のTCPの最大速度を制限します。
- **モーメント**：ロボット動作時の全体のモーメントの最大値を制限します。
- **停止時間**：ロボットのアラームが発生した際（非常停止を含む）の停止に要する最大時間を制限します。
- **停止距離**：ロボットのアラームが発生した際（非常停止を含む）の停止に要する最大距離を制限します。

ロボットの動作計画では、上記のパラメータ値が制限範囲内に収まるよう、実際の状況に応じて計画された最大速度を調整します。この条件下で、ロボットの動作中に **TCP力** または **パワー** が制限値を超えた場合、ロボットが衝突したと見なされ、自動的に停止して衝突検出アラームが作動します。

> **説明:**
> TCP速度、モーメント、停止時間、停止距離は、ロボットの最大TCP速度を制限します。パラメータの値が大きいほど、ロボットの動作速度が高くなります。

ジョグ操作中と自動運転中にロボットが衝突した場合の対応方法は異なります。

**ジョグ操作中に衝突が発生した場合**

ソフトウェアに衝突検出のポップアップ通知が表示されます。この場合は、衝突の原因を解決し、**リセット** をクリックしてください。衝突の原因を解決する際に制御ソフトウェアを操作する必要がある場合、**1分後に再通知** をクリックしてポップアップを一時的に閉じることができます（1分後に再度通知が表示されます）。


**自動運転中に衝突が発生した場合**

- ロボットは設定された**衝突後の処理方法**に従って動作します：
  - **停止**：ロボットが動作を停止します。
  - **一時停止**：ロボットが一時停止します。状況に応じて、衝突原因を解決して続行するか、停止するかを選択します。ロボットが衝突一時停止状態にある場合、ロボットのエ

ンドエフェクタにあるドラッグティーチボタンを短押しすることで動作を再開することもできます。

どの処理方法を選択した場合でも、ロボットが衝突した後は、衝突前の軌跡に沿って指定された距離だけ自動で後退します。後退距離の設定範囲は0～50mmで、デフォルト値は10mmです。

**編集** をクリックするとパラメータを変更できます。変更が完了したら **保存** をクリックして変更を保存するか、**キャンセル** をクリックして今回の変更を取り消します。**デフォルト値に戻す** をクリックすると、パラメータを出荷時のデフォルト値にリセットできます。

**説明:**
縮減モードでの値は必ず通常モードの値より小さく設定してください。そうでない場合、保存できません。

# 10.13.3 関節リミット（CRAシリーズ）

ユーザーは、ロボットの各関節のソフトリミットを設定することで、各関節の動作範囲を制限することができます。

ロボットの実際の関節角度が（負リミット + 負リミット許容範囲）より小さい場合、または（正リミット + 正リミット許容範囲）を超えた場合、アラームが作動し、動作が停止します。

負リミットと正リミットの値は変更可能です。

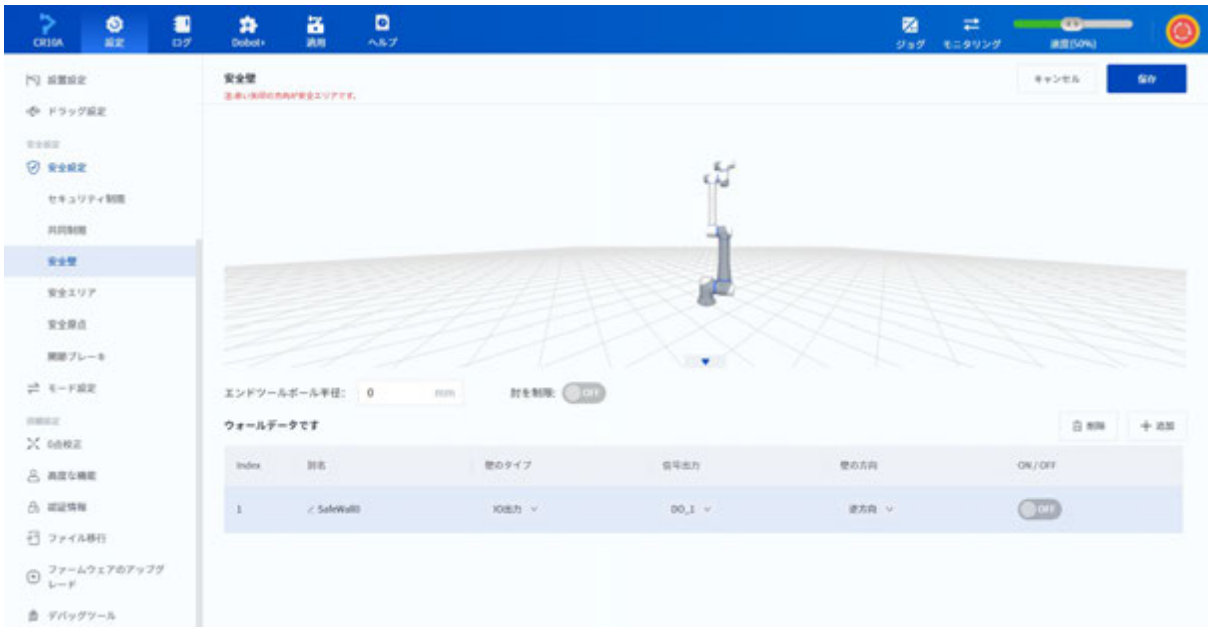
**編集** をクリックするとパラメータを変更できます。変更が完了したら **保存** をクリックして変更を保存するか、**キャンセル** をクリックして今回の変更を取り消します。**デフォルト値に戻す** をクリックすると、パラメータを出荷時のデフォルト値にリセットできます。

# 10.13.4 安全壁

DobotStudio Proは最大8つの安全壁を設定することができます。壁の方向によりロボットの安全空間を指定することができます。ロボットの**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）**が安全壁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。

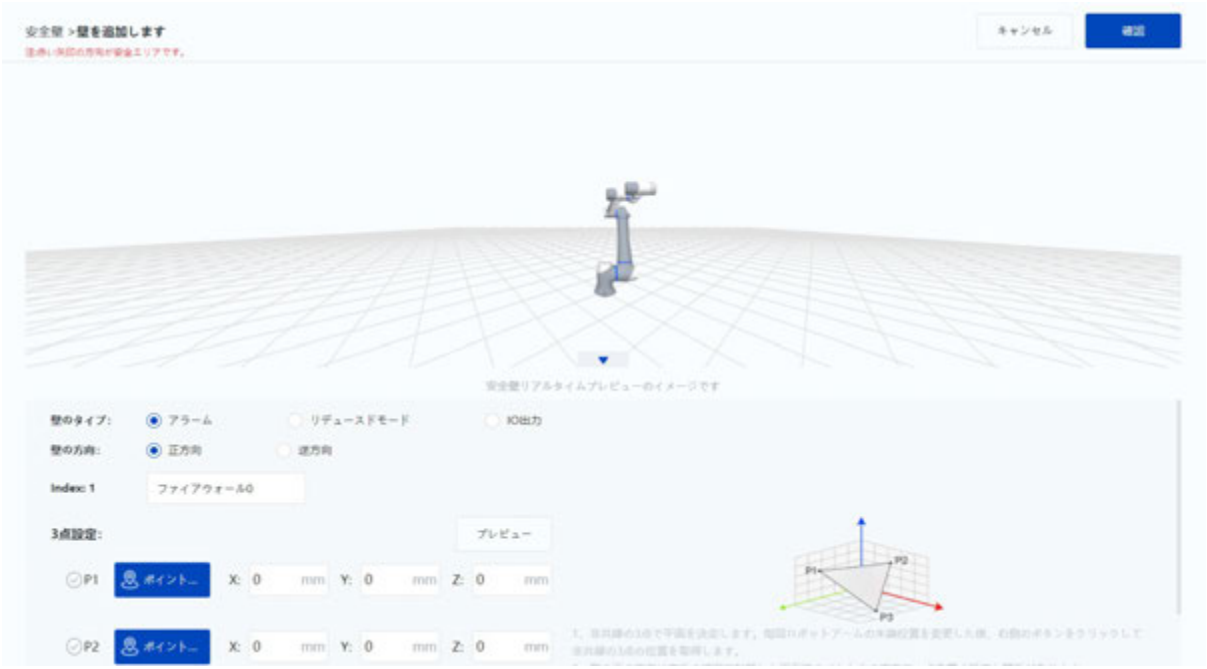
- **アラーム**：安全境界に近づいたとき、現在の動作計画によってロボットが安全範囲を超えると、アラームがトリガされ、動作が停止します。
- **縮減**：安全領域を超えると、ロボットの減速モードがトリガされ、減速します。
- **IO出力**：安全領域を超えると、ロボットは指定されたDOをトリガし、動作には影響しません。

異なる安全エリアまたは安全壁は同時に有効であり、同じDOを設定すると、有効な安全エリアまたは安全壁をトリガすると、対応する動作がトリガされます。




## 安全壁の追加

＋ **追加** をクリックして、新しい安全エリアを追加することができます。安全壁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。ユーザーは先に3つの点を決める必要があります。以下P1、P2とP3と言います。




1. ジョグまたはドラッグでロボットをPoint1まで移動し、**ポイント取得** をクリックしてPoint1の座標を取得します。
2. 同じ方法でPoint2とPoint3の座標を取得します。**プレビュー**をクリックすると、生成された安全エリアは上のシミュレーションエリアで確認することができます。

> **説明:**
> - ポイントのデカルト座標を手動で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、シミュレーションエリアの表示が更新されます。

3. 安全壁のタイプを設定します。
4. 安全壁の方向を設定します。壁の方向はシミュレーションエリアで確認することができます。矢印の方向が壁の安全側、その逆の方向が制限側になります。

> **説明:**
> 壁の正の方向は、P1 -> P2 -> P3 のベクトル方向に基づいて右手の法則に従って計算される平面法線ベクトルとなります。

5. **確認**をクリックして、安全壁を追加します。

## 安全壁の変更


**Index** を除く安全壁の属性はすべて変更可能です。ただし、**信号出力** は**ウォールタイプ**が**IO出力**の場合のみ設定できます。右側のスイッチで安全壁の有効/無効を切り替えることができます（ロボットがディスエーブル状態の場合のみ操作可能）。有効化された安全ウォールのみがロボットアームと干渉し、シミュレーションエリアに表示されます。

安全壁を選択した状態で 削除 **削除** をクリックすると、選択した安全ウォールを削除できます。

## 高度な設定


**エンドツール球の半径**

エンドツール球（TCPを球の中心とするカスタマイズ可能な半径の球形空間）の半径を指定します。この球形空間が安全ウォールの制限エリアと干渉した場合、安全ウォールのアクションがトリガーされます。0に設定すると、TCPのみが制限エリアと干渉した場合にアクションがトリガーされます。

**肘部の制限**

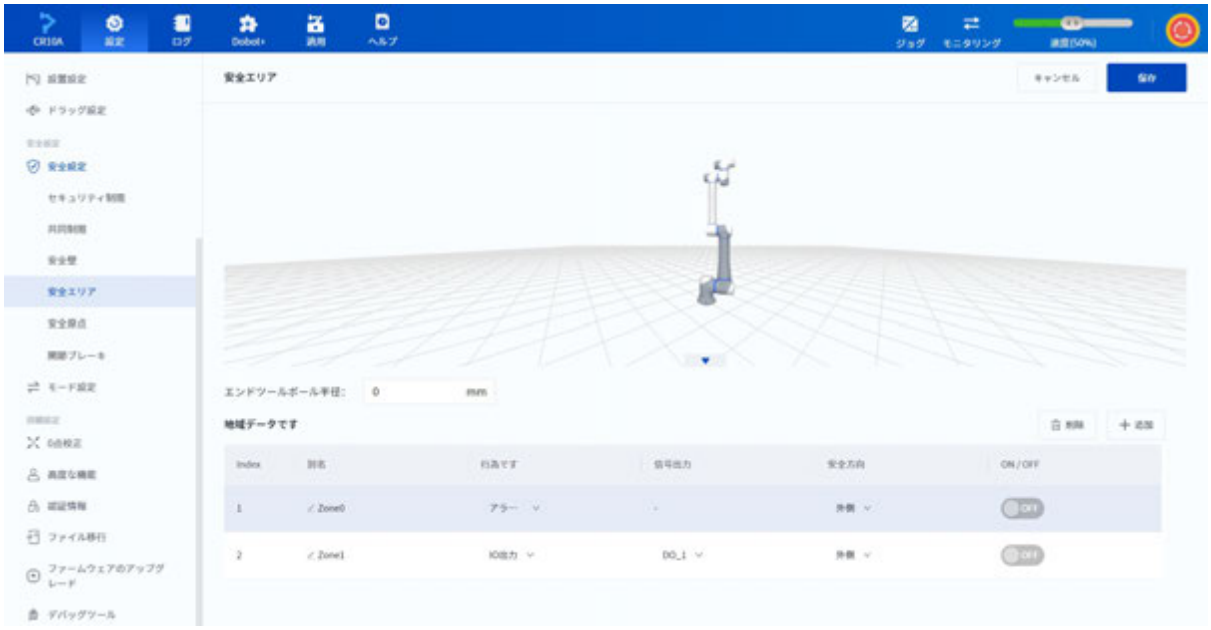
このオプションを **ON** に設定すると、J3関節も制限エリアと干渉するようになります。**OFF** に設定した場合は、エンドツール球のみが制限エリアと干渉します。

# 10.13.5 安全エリア

DobotStudio Proは、最大6つの安全エリアを設定することが可能です。指定したエリアの内側または外側をロボットの安全エリアとして定義します。ロボットの**エンドツール球（TCPを球の中心とし、カスタマイズ可能な半径の球形空間）**が安全エリアに接近または超えた際、安全エリアの動作タイプに応じて異なるアクションがトリガーされます。

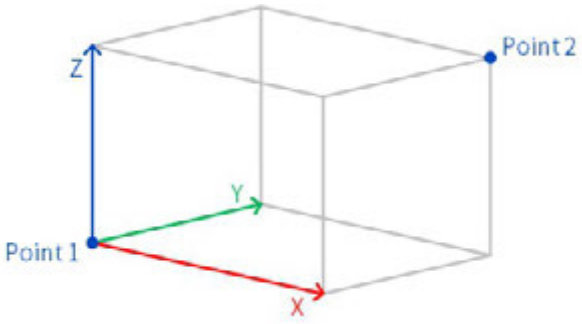
- **アラーム**：安全境界に接近し、現在の動作計画に従うとロボットが安全範囲を超える場合、アラームがトリガーされ動作が停止します。
- **縮減**：安全空間を超えた場合、ロボットは縮減モードをトリガーし、減速して動作します。
- **IO出力**：安全空間を超えた場合、ロボットが指定されたDOをトリガーしますが、動作には影響しません。

異なる安全エリアや安全壁が同時に有効となる場合、それぞれに同じDOを設定することが可能です。有効な安全エリアまたは安全壁のいずれかがトリガーされると、対応するアクションが実行されます。




## 安全エリアの追加

＋ **追加** をクリックして、新しい安全エリア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、安全エリアの対角の2つの頂点 (Point1 と Point2) をティーチンし、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。下図の通りです。





1. ジョグまたはドラッグでロボットをPoint1まで移動し、**ポイント取得** をクリックしてPoint1の座標を取得します。
2. 同じ方法で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、生成された安全エリアは上のシミュレーションエリアで確認することができます。

   > **説明:**
   > - ポイントのデカルト座標を手動で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、シミュレーションエリア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エリアを超えた場合の動作を設定します。
5. 安全方向を設定します。**外側**は立方体の外側のエリアが安全エリアであることを示します。**内側**は立方体の内部が安全エリアであることを示します。
6. **確認**をクリックして、安全エリアを追加します。

## 安全エリアの変更


安全エリアは**Index**と**領域形状**以外のプロパティは変更することができます。**信号出力**は**アクション**が**IO出力**である安全エリアだけが設定可能です。右側のスイッチで安全エリアを有効にするかどうかを制御できます（ロボットがディセーブ状態でのみ操作可能）。有効な安全エリアがロボットと干渉すると、右側のシミュレーションエリアに表示されます。

安全エリアを選択し、**×削除**をクリックすると、選択した安全エリアを削除することができます。

## 高度な設定


**エンドツール球の半径**

エンドツール球（TCPを球の中心とするカスタマイズ可能な半径の球形空間）の半径を指定します。この球形空間が非安全エリアと干渉した場合、安全エリアのアクションがトリガーされます。0に設定すると、TCPのみが制限エリアと干渉する場合に限りアクションがトリガーされます。

# 10.13.6 安全原点

安全原点はカスタマイズ可能な姿勢です。デフォルトの姿勢はゼロ点姿勢、つまり各関節の角度が0です。現場の用途に応じて変更することを推奨します。ロボットが安全原点にある場合、安全I/O、システムI/O、または Modbusを介して安全原点状態信号を出力することができ、ユーザーはその信号に基づいて判断し、ロボットが安全な原点にある場合にのみプロジェクトを実行したりすることができます。


- **移動**を長押しすると、ロボットを安全原点に移動させることができます。
- **安全原点をリセット**をクリックすると、安全原点を変更することができます。

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**ポイント取得**をクリックし、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。**デフォルトポイントに復元**をクリックすると、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、**保存**をクリックすると、安全原点が更新されます。

# 10.13.7 関節ブレーキ

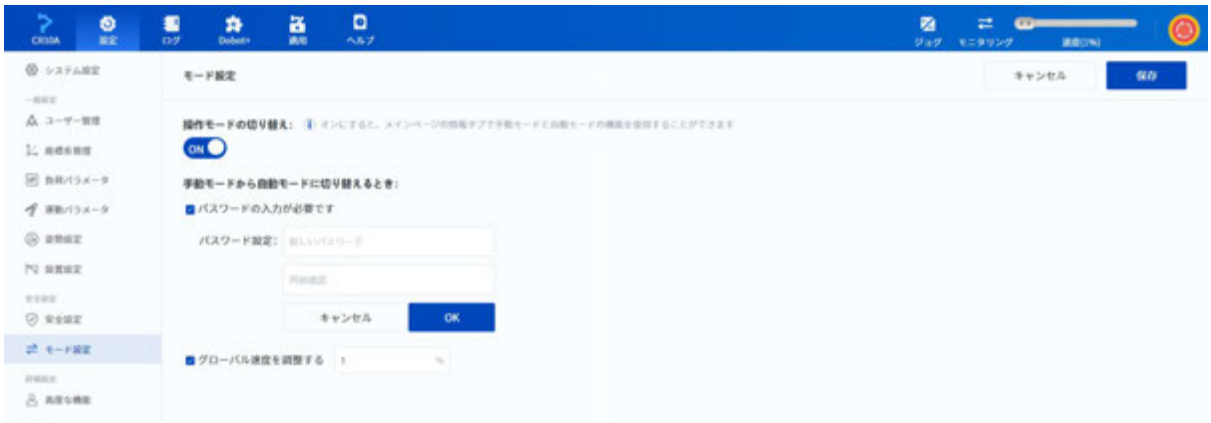
ロボットがディセーブル状態の時、関節の移動を防止するため、関節は自動的にブレーキをかけてモータの位置をロックし、機械の可動部分が自重や外力で移動しないようにします。緊急の場合、ユーザーはこの画面から関節ブレーキを解除し、対応する関節を手でドラッグして移動することができます。


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。

# 10.14 モード設定

ロボットの手動/自動モード切替機能は出荷時にデフォルトで無効になっています（CR20は出荷時に手動モードがデフォルト）。このページで有効にすることができます。手動/自動モードは、現場での安全性を向上させるための機能であり、リスク評価の結果に基づいて有効化するかどうかを選択してください。




**操作モード切替** 機能を有効にした場合、手動モードから自動モードに切り替える際に **パスワードの入力が必要** かどうかを選択できます。このオプションを有効にした場合、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）。

**全体速度調整** 機能を選択した場合、自動モードに切り替えると全体速度が設定した値に自動的に調整されます。**全体速度調整** では、1～100の整数値のみを入力できます。

# 10.15 0点校正

ロボットのモーターや減速機などの伝達部品を交換したり、ワークに衝突したりすると、ロボットのゼロ点位置ズレが発生します。この際、ロボットのゼロ点キャリブレーションが必要になります。


ページの指示に従い、まずロボットアームをゼロ点姿勢に移動させます（ロボットアームの各関節に貼付されたゼロ点ステッカーを使用してキャリブレーションを行います。詳細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参照してください）。

その後、ロボットがイネーブル状態で、ページ右上の **すべての関節をキャリブレーション** ボタンをクリックしてすべての関節をキャリブレーションするか、各関節の対応する **ゼロ点キャリブレーション** ボタンをクリックして単一の関節をキャリブレーションします。

**説明:**
ゼロ点キャリブレーションは、ゼロ点位置が変更された場合にのみ使用してください。操作には十分注意してください。

キャリブレーションが成功した後、ロボットのイネーブルを解除する必要があります。この操作によってキャリブレーションが有効になります。その後、コントロールパネルで関節座標を確認できます。この時点で、J1～J6の値はすべて0になります。

# 10.16 高度な機能

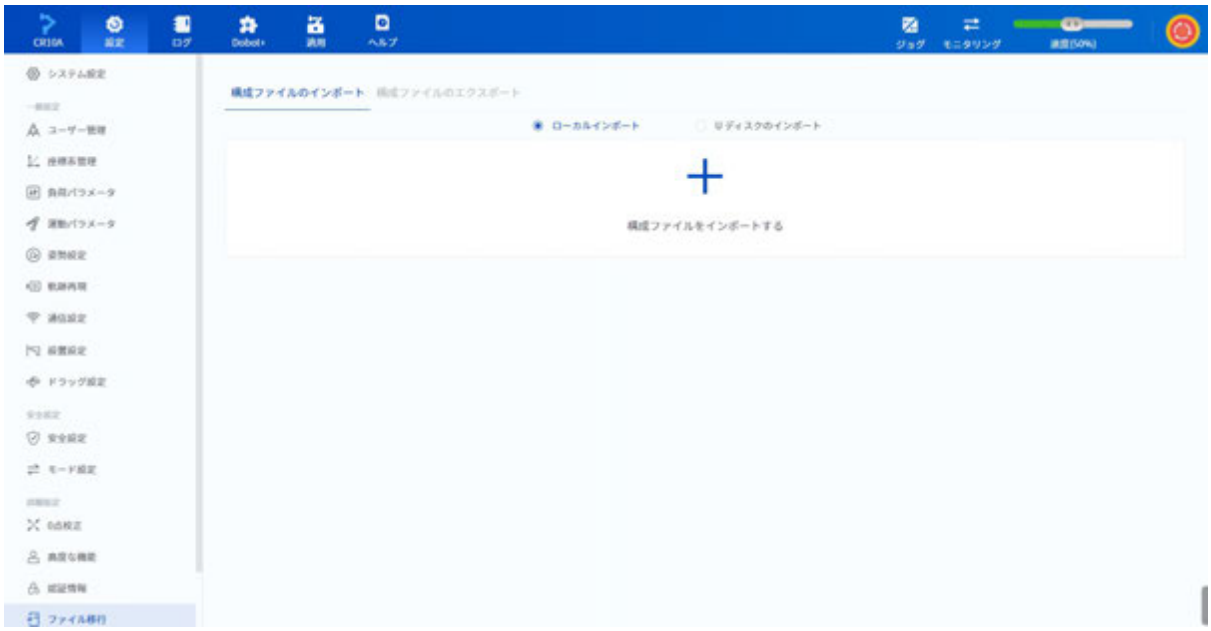
現場の状況により高度機能をオン/オフする場合は、この画面で設定することができます。特別な要件がない場合は、デフォルト値を保持することを推奨します。

**説明:**
**一時停止状態では、ジョグ操作**および**軌跡復元**機能がサポートされています。詳細については、**軌跡復元**をご参照ください。

# 10.17 ファイル移行

DobotStudio Proは、ロボット内部ファイル（プログラムファイル、設定ファイルなど）のインポートおよびエクスポート機能をサポートしています。この機能は、システムバックアップやデバイスのコピーなどの場面で使用でき、ローカルまたはUSBメモリを介してインポートおよびエクスポートが可能です。

## 設定ファイルのインポート

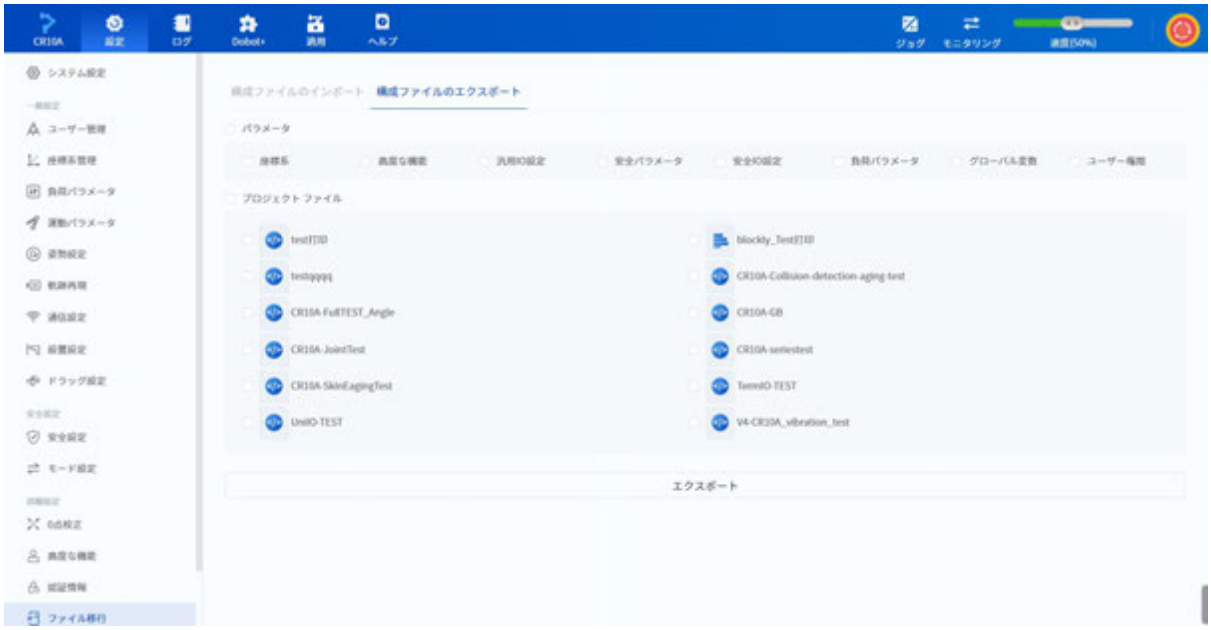


「**設定** > **ファイル移行**」 をクリックして 「**設定ファイルのインポート**」 ページに進みます。「**ローカルインポート**」 または 「**USBインポート**」 を使用して、ローカルまたはUSBメモリに保存されている設定ファイルをシステムにインポートすることができます。

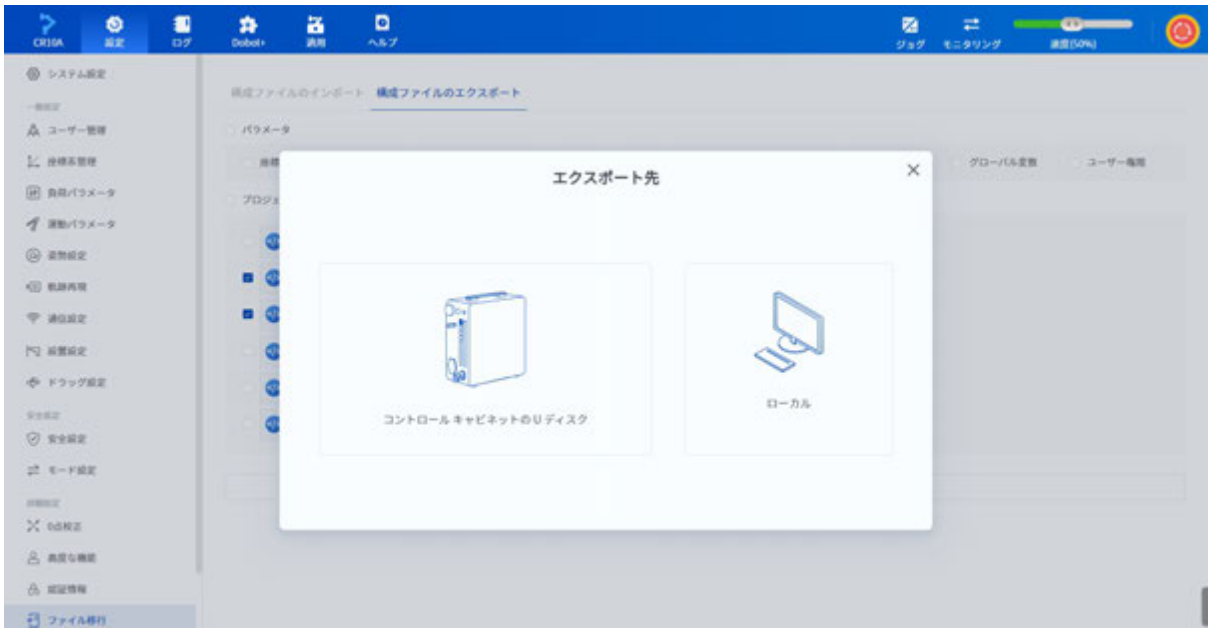
**説明:**

- 設定ファイルをインポートする際は、ロボットアームをディセーブ状態にしてから操作を続行してください。
- USBインポートでは、USBメモリのルートディレクトリにある圧縮ファイルのみが表示されます。正しいファイルを選択してください。

## 設定ファイルのエクスポート



「**設定ファイルのエクスポート**」 タブをクリックし、エクスポートするパラメータとプロジェクトファイルを選択して、「**エクスポート**」 ボタンをクリックします。その後、「**保存先**」 の確認ウィンドウが表示されます。



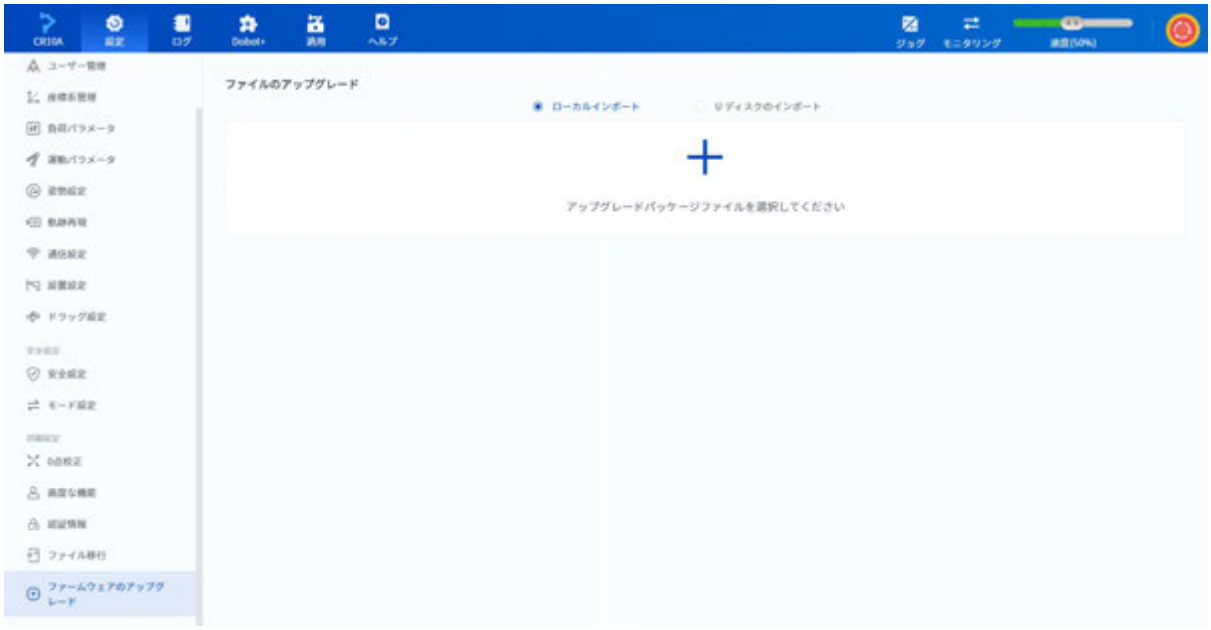

システム設定ファイルを 「**コントローラーのUSBメモリ**」 または 「**ローカル**」 にエクスポートすることを選択します。

**説明:**

- 「コントローラーのUSBメモリ」 にエクスポートを選択した場合、デフォルトでUSBメモリのルートディレクトリにエクスポートされます。
- コントローラーのUSBメモリを使用してインポートまたはエクスポートする際は、コントローラーの任意のUSBポートにUSBメモリを差し込むことができます。USBポートに2つのUSBメモリが接続されている場合は、最初に検出されたUSBメモリのフォルダが使用されます。
- エクスポートできるプロジェクト数は100件以内に制限されています。

# 10.18 ファームウェアアップグレード

「**ファームウェア更新**」ページを使用して、ロボットのファームウェアをワンクリックで最新バージョンにアップデートしたり、ファームウェアのバージョンをロールバックしたりすることができます。



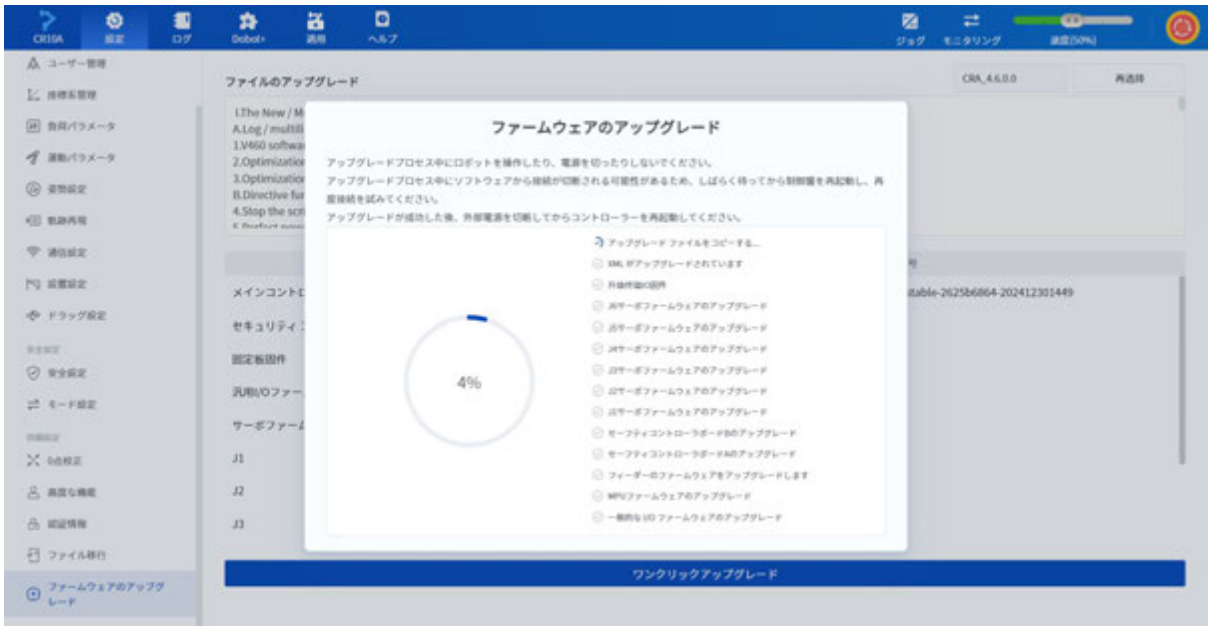

「**設定** > **ファームウェア更新**」 をクリックすると、「**ローカルインポート**」 または 「**USBインポート**」 を通じて、ローカルまたはUSBメモリに保存されているアップデートパッケージファイルをシステムにインポートすることができます。

**説明:**

- アップデートパッケージファイルをインポートする際は、ロボットアームをディスエーブル状態にしてから操作を続行してください。アップデート中にロボットアームの電源を切らないでください。
- USBインポートを使用してファームウェアを更新する場合、ネットワークポートのないUSBハブにUSBメモリを接続することを推奨します。
- USBインポート時、USBメモリのルートディレクトリにある圧縮ファイルのみが表示されます。正しいファイルを選択してください。
- ファームウェア更新を使用できるのは管理者権限のみです。この権限は変更できません。

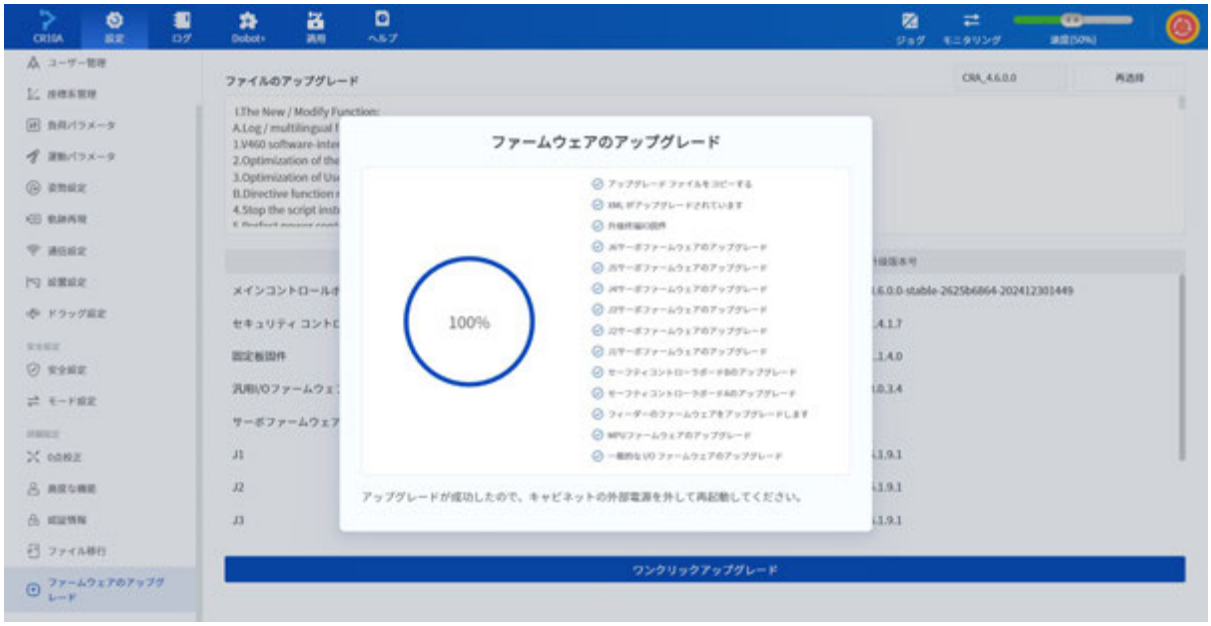
パッケージファイルをインポートすると、ファイルがローカルに解凍されます。解凍に成功すると、新旧のファームウェアバージョン情報とアップデート内容が表示されます。


「**ワンクリックアップグレード**」 をクリックすると、バージョンのアップデートが開始されます。画面にはアップデートの進行状況が表示され、この間は他の機能を使用することはできません。




- アップグレード中に「アップグレードファイルのコピーに失敗しました」と表示された場合は、「**再アップグレード**」 をクリックしてワンクリックアップグレードを再試行してください。
- アップグレード中に「アップグレードファイルのコピーに失敗しました」以外のエラーメッセージが表示された場合は、インターフェースから返されたエラー情報に基づいて問題を解決してください。
- アップグレード完了後に「アップグレード成功」と表示された場合は、ロッカスイッチを操

作してコントローラーの電源を切り、再起動してください。コントローラーを再起動後、ロボットをDobotStudio Proに再接続してください。




**説明:**
ファームウェアのバージョンをアップグレードする際は、ロボットメンテナンスツールを優先的に使用してください。

# 付録A Modbusレジスタ定義

## 1 Modbus説明

Modbusはシリアル通信プロトコルの一種で、このプロトコルを通じてロボットは外部デバイスと通信できます。PLCなどの外部デバイスを使用してロボットを制御する場合、外部デバイスはModbusのマスターデバイスとして、ロボットはModbusのスレーブデバイスとして動作します。

Modbusの主な種類は以下の通りです:

- **Modbus-RTU**: コンパクトで、データをバイナリ形式で表現します。巡回冗長検査（CRC）方式でチェックサムを使用します。
- **Modbus-ASCII**: ASCIIコードに基づく文字列形式で表現され、読みやすく冗長な形式です。縦方向冗長検査（LRC）方式でチェックサムを使用します。
- **Modbus-TCP**: TCP/IPのTCP方式を通じて接続します。この方式ではチェックサムと計算が不要です。
- **Modbus-RTU-over-TCP**: TCPで転送されるRTUコンテンツのデータパケットです。

現在、当社はModbus-TCPおよびModbus-RTU-over-TCPの方式を提供しています。

Modbusプロトコルに基づき、Dobotの6軸協働ロボットに割り当てられるModbusアドレスは以下の通りです:

- **00001-09999**: コイルレジスタ。値は0または1のみ対応可能で、ロボットの制御に使用され、読取・書込が可能です。
- **10001-19999**: 接点レジスタ。値は0または1のみ対応可能で、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。最大64ビットの倍精度浮動小数点数に対応し、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。最大64ビットの倍精度浮動小数点数に対応し、ロボットとPLCの相互作用に使用され、読取・書込が可能です。

各種レジスタに対応するModbus機能コードは、標準Modbusプロトコルに準拠しています。

| レジスタタイプ | レジスタの読み取り | 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 |

ロボットは外部デバイスによるアクセス用に2つのテーブルを提供しています。map1は502（Modbus-TCP）および503（RTU-over-TCP）ポートを介してアクセス可能です。map2は1502（Modbus-TCP）および1503（RTU-over-TCP）ポートを介してアクセス可能です。それぞれのリソースの詳細は以下の通りです：

| パラメータ/スレーブ | 502 | 503 | 1502 | 1503 |
|---|---|---|---|---|
| コイル数 | 10000 | 10000 | 10000 | 10000 |
| 離散入力数 | 10000 | 10000 | 10000 | 10000 |
| 入力レジスタ数 | 10000 | 10000 | 10000 | 10000 |
| 保持レジスタ数 | 10000 | 10000 | 10000 | 10000 |
| 所属マップ | map1 | map1 | map2 | map2 |
| バインドポート | 502 | 503 | 1502 | 1503 |
| Modbusプロトコル形式 | TCP | RTU-over-TCP | TCP | RTU-over-TCP |
| サポートする最大マスター数 | 20 | 20 | 20 | 20 |
| スレーブID | 1 | 1 | 1 | 1 |

なお、map1の一部のアドレスはロボットシステムによってデフォルトで使用されています。詳細については、以下のレジスタ定義をご参照ください。map2は現在空の状態であり、ユーザーが実際のニーズに応じて使用できます。

# 2 コイルレジスタ（map1、ロボット制御）

| PLC地址 | 脚本地址（Get/SetCoils） | 寄存器类型 | 功能 |
|---|---|---|---|
| 00001 | 0 | Bit | 开始 |
| 00002 | 1 | Bit | 停止 |
| 00003 | 2 | Bit | 暂停 |
| 00004 | 3 | Bit | 上使能 |
| 00005 | 4 | Bit | 下使能 |
| 00006 | 5 | Bit | 清除报警 |
| 00007 | 6 | Bit | 进入拖拽 |
| 00008 | 7 | Bit | 退出拖拽 |
| 00009 | 8 | Bit | 切换至自动模式 |
| 00010 | 9 | Bit | 切换至手动模式 |
| | | | |

| | | | |
|---|---|---|---|
| 00011~01024 | 10~1023 | Bit | 保留位 |
| 01025~09999 | 1024~9998 | Bit | 用户自定义 |

# 3 接点レジスタ定義（map1、ロボットステータス）

| 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 | 実行状態 |
| 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 | セーフスキン一時停止状態 |
| 10006 | 5 | Bit | 待機状態 |
| 10007 | 6 | Bit | 通電状態 |
| 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 | ドラック状態 |
| 10012 | 11 | Bit | リカバリモード状態 |
| 10013~19999 | 12~9998 | 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（32001~32067を除く。このアドレスにはU16タイプのコントローラーSNコード、ロボットSNコード、およびセーフティチェックサムの文字が保存されています）。バイトオーダーはリトルエンディアン形式で反転されています。

| PLCアドレス | データタイプ | 点数 | 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 | メッセージ長 |
| 31002~31004 | - | - | - | 1001~1003 | - | 保留 |
| 31005~31008 | uint64 | 1 | 8 | 1004~1007 | U64 | デジタル入力、詳細はDI/DOの説明にご参照ください |
| 31009~31012 | uint64 | 1 | 8 | 1008~1011 | U64 | デジタル出力、詳細はDI/DOの説明にご参照ください |
| 31013~31016 | uint64 | 1 | 8 | 1012~1015 | U64 | ロボットモード、詳細はRobotModeの説明をご参照ください |
| 31017~31020 | uint64 | 1 | 8 | 1016~1019 | U64 | Unixタイムスタンプ（単位ms） |
| 31021~31024 | - | - | - | 1020~1023 | - | 保留 |
| 31025~31028 | uint64 | 1 | 8 | 1024~1027 | U64 | メモリ構造試験基準値 0x0123 4567 89AB CDEF |
| 31029~31032 | - | - | - | 1028~1031 | - | 保留 |
| 31033~31036 | double | 1 | 8 | 1032~1035 | F64 | 速度比率 |
| 31037~3104 | - | - | - | 1036~1039 | - | 保留 |
| 31041~31044 | double | 1 | 8 | 1040~1043 | F64 | コントロールボード電圧 |
| 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 | - | 保留 |
| 31097~31120 | double | 6 | 48 | 1096~1119 | F64 | 目標の関節位置 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 31121~31144 | double | 6 | 48 | 1120~1143 | F64 | 目標の関節速度 |
| 31145~31168 | double | 6 | 48 | 1144~1167 | F64 | 目標の関節加速度 |
| 31169~31192 | double | 6 | 48 | 1168~1191 | F64 | 目標の関節電流 |
| 31193~31216 | double | 6 | 48 | 1192~1215 | F64 | 目標の関節トルク |
| 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センサのトルク値<br>(6軸力覚による計算) |
| 31313~31336 | double | 6 | 48 | 1312~1335 | F64 | TCPデカルト実際の座標値 |
| 31337~31360 | double | 6 | 48 | 1336~1359 | F64 | TCPデカルト実際の速度値 |
| 31361~31384 | double | 6 | 48 | 1360~1383 | F64 | TCPトルク値<br>(関節電流による計算) |
| 31384~31408 | double | 6 | 48 | 1384~1407 | F64 | TCPデカルト目標の座標値 |
| 31409~31432 | double | 6 | 48 | 1408~1431 | F64 | TCPデカルト目標の速度値 |
| 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。<br>8：位置モード;<br>10：トルクモード |
| 31481~31504 | double | 6 | 48 | 1480~1503 | F64 | 関節電圧 |
| 31505~31506 | - | - | - | 1504~1505 | - | 保留 |
| | | | | 1506下位バ | | ユーザー座標 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | イト | | 系 |
| 31507 | char | 1 | 1 | 1506上位バイト | I8 | ツール座標系 |
| 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 | デカルトポーズ速度比 |
| 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 | デカルトポーズ加速度比 |
| 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 | デカルトポーズ加加速度比 |
| 31513 | char | 1 | 1 | 1512上位バイト | I8 | ロボットブレーキ状態、詳細はBrakeStatusの説明にご参照ください |
| 31514 | char | 1 | 1 | 1513下位バイト | I8 | ロボットイネーブル状態 |
| 31514 | char | 1 | 1 | 1513上位バイト | I8 | ロボットドラッグ状態 |
| 31515 | char | 1 | 1 | 1514下位バイト | I8 | ロボット実行状態 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 31515 | char | 1 | 1 | 1514上位バイト | I8 | ロボットアラーム状態 |
| 31516 | char | 1 | 1 | 1515下位バイト | I8 | ロボットジョグ状態 |
| 31516 | char | 1 | 1 | 1515上位バイト | I8 | ロボットのタイプ、詳細はRobotTypeの説明にご参照ください |
| 31517 | char | 1 | 1 | 1516下位バイト | I8 | エンドパネルのドラッグ信号 |
| 31517 | char | 1 | 1 | 1516上位バイト | I8 | エンドパネルのイネーブル信号 |
| 31518 | char | 1 | 1 | 1517下位バイト | I8 | エンドパネルの録画信号 |
| 31518 | char | 1 | 1 | 1517上位バイト | I8 | エンドパネルの再現信号 |
| 31519 | char | 1 | 1 | 1518下位バイト | I8 | エンドパネルのハンド制御信号 |
| 31519 | char | 1 | 1 | 1518上位バイト | I8 | 6軸力覚オンライン状態 |
| 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 | (セフティスキン)アッパーアーム接近一時停止状態 |
| 31521 | char | 1 | 1 | 1520下位バイト | I8 | (セフティスキン)J4軸接近一時停止状態 |
| 31521 | char | 1 | 1 | 1520上位バイト | I8 | (セフティスキン)J5軸接近一時停止状態 |
| | | | | | | (セフティス |

| | | | | | | |
|---|---|---|---|---|---|---|
| 31522 | char | 1 | 1 | イト | I8 | J6軸接近一時停止状態 |
| 31522 | char | 1 | 1 | 1521上位バイト | I8 | 保留 |
| 31523~31552 | - | - | - | 1522~1551 | - | 保留 |
| 31553~31556 | double | 1 | 8 | 1552~1555 | F64 | 保留 |
| 31557~31560 | uint64 | 1 | 8 | 1556~1559 | U64 | 現在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方向オフセット距離（mm） |
| 31593~31596 | double | 1 | 8 | 1592~1595 | F64 | Y方向オフセット距離（mm） |
| 31597~31600 | double | 1 | 8 | 1596~1599 | F64 | Z方向オフセット距離（mm） |
| 31601~31624 | double | 6 | 48 | 1600~1623 | F64 | ユーザー座標値 |
| 31625~31648 | double | 6 | 48 | 1624~1647 | F64 | ツール座標値 |
| 31649~31652 | double | 1 | 8 | 1648~1651 | F64 | 軌道再現実行索引 |
| 31653~31676 | double | 6 | 48 | 1652~1675 | F64 | 現在6軸力覚元データ値 |
| 31677~31692 | double | 4 | 32 | 1676~1691 | F64 | [qw,qx,qy,qz]目標クオータニオン |
| 31693~31708 | double | 4 | 32 | 1692~1707 | F64 | [qw,qx,qy,qz]実際クオータニオン |
| 31709 | unsigned short | 1 | 2 | 1708 | U16 | 手動/自動状態。<br>1：手動<br>2：自動モード<br>0：モード切替無効 |
| 31710 | unsigned short | 1 | 2 | 1709 | U16 | USBエクスポート状態 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 31711 | char | 1 | 1 | 1710下位バイト | I8 | セーフティ状態 |
| 31711 | char | 1 | 1 | 1710上位バイト | I8 | セーフティ状態予約ビット |
| 31710~31721 | double | 1 | 24 | 1709~1720 | F64 | 保留 |
| | | | 1440 | | | 計1440バイト |

| PLCアドレス | スクリプトアドレス (GetInRegs) | タイプ | 機能 |
|---|---|---|---|
| 32001~32032 | 2000~2031 | U16 | コントローラーSNコードの第1文字目<br>~第32文字目 |
| 32033~32065 | 2032~2064 | U16 | ロボットSNコードの第1文字目<br>~第32文字目 |
| 32066 | 2065 | U16 | 「セーフティチェックサム」の下位4文字<br>(2バイト) |
| 32067 | 2066 | U16 | 「セーフティチェックサム」の上位4文字<br>(2バイト) |

**モーションパラメータフィードバック値の説明**

モーションパラメータ（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ず、次のモーションコマンドを実行すると更新されます。

**DI/DOの説明**

DI/DO はそれぞれ 8 バイトを占め、各バイトは 8 ビット (バイナリ) で構成されて、最大 64 個の DI/DO ポートの状態を表すことができます。各バイトのLowからHighまでの各ビットは端子の状態を表し、1は対応する端子がONであることを意味し、0は対応する端子がOFFであること、または対応する端子が存在しないことを意味します。

例えば、DI の最初のバイトは 0x01で、バイナリ表現は00000001です。LowからHighまで、D1_1 ~ DI_8のステータスを示します。つまり、DI_1がON、残りの7つのDIがOFFです。

2番目のバイトは 0x02で、バイナリ表現は00000010です。LowからHighまで、D1_9 ~ DI_16のステータスを示します。つまり、DI_10がON、残りの7つのDIがOFFです。

以降のバイトも同様です。制御盤によってはIO端子数が異なりますので、IO端子数を超えるバイナリビットはすべて0となります。

**RobotModeの説明**

| 設定値 | 定義 | 説明 |
|---|---|---|
| 1 | ROBOT_MODE_INIT | 初期化状態 |
| 2 | ROBOT_MODE_BRAKE_OPEN | どれかの関節ブレーキが解除された |
| 3 | ROBOT_MODE_POWEROFF | ロボット電源OFF状態 |
| 4 | ROBOT_MODE_DISABLED | ディセーブル（ブレーキ解除なし） |
| 5 | ROBOT_MODE_ENABLE | イネーブルかつ待機 |
| 6 | ROBOT_MODE_BACKDRIVE | ドラッグモード |
| 7 | ROBOT_MODE_RUNNING | 実行状態（プロジェクト、TCPキュー実行など） |
| 8 | ROBOT_MODE_SINGLE_MOVE | シングルモーション状態（ジョグ、RunToなど） |
| 9 | ROBOT_MODE_ERROR | 未処理のアラームがあります。この状態が最も優先されます。ロボットがどのような状態であっても、アラームがある場合は9を返します。 |
| 10 | ROBOT_MODE_PAUSE | プロジェクト一時停止状態 |
| 11 | ROBOT_MODE_COLLISION | 衝突検知状態 |

**BrakeStatus説明**

このバイトは各関節のブレーキ状態をビットごとに表します。対応するビットが1の場合、その関節のブレーキが解除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以下の表の通りです:

| ビット | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **意味** | 保留 | 保留 | 関節1 | 関節2 | 関節3 | 関節4 | 関節5 | 関節6 |

示例:

- 0x01（00000001）：関節6ブレーキ解除
- 0x02（00000010）：関節5ブレーキ解除
- 0x03（00000011）：関節5と関節6ブレーキ解除

- 0x04（00000100）：関節4ブレーキ解除

**RobotType説明**

| 設定値 | 代表機種 |
|---|---|
| 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，ロボットとPLC間の通信）

| 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 | - | 保留 |
| 41025~49999 | 1024~9998 | - | カスタム |

# 付録B Blocklyプログラミングブロックの説明


- B.1 共通説明
- B.2 イベントブロックグループ
- B.3 制御ブロックグループ
- B.4 演算ブロックグループ
- B.5 文字列ブロックグループ
- B.6 カスタムブロックグループ
- B.7 IOブロックグループ
- B.8 モーションブロックグループ
- B.9 Modbusブロックグループ
- B.10 バスブロックグループ
- B.11 TCPブロックグループ
- B.12 トレイブロックグループ
- B.13 クイックスタート

# 共通説明

## ブロックタイプ

ブロックプログラミングのブロックは、形状によって4つに分類されています。

**丸形ブロック**


この形状のブロックは、ブロック列の先頭で使用されます。実行されるすべてのブロックは、このタイプ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にのみ含まれます。

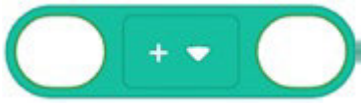
**矩形ブロック**


この形状のブロックはコマンドを実行するために使用されます。ブロックプログラミングムの主な構成です。矩形ブロックの中にある楕円形及びおよび菱形のパラメータスロットは、対応する形状のブロックで埋め込むことができます。


矩形ブロックの1つのバリエーションとして、ブロックの中に他のブロックを入れ子にすることができ、フロー制御に使用されます。

例:


**楕円形ブロック**




楕円形ブロックを実行すると、変数（数値/文字列/配列など）が返されます。この変数は、矩形ブロックの対応する形状のパラメータスロットに挿入し、パラメータとして使用します。

例:


**菱形ブロック**


この形状のブロックは条件判断として使用されます。実行した後、結果をtrue或いはfalseとして返し、矩形ブロックの中にある対応する形状のパラメータースロットに菱形ブロックで埋めることができます。

例:


## 運動方式

ロボットアームがサポートする運動方式は以下の種類に分類されます。

**関節運動**

ロボットアームは現在点の関節角度と目標点の関節角度の差に基づいて、各関節が同時に動作を完了するように計画します。関節動作はTCP（Tool Center Point）の動作軌道を制約しません。一般的には動作軌道は非直線となります。


関節動作は特異点の位置制限を受けません（特異点の位置についてはロボットアームの該当するハードウェアマニュアルを参照してください）。そのため、動作軌跡に特に要件がない場合や、目標地点が特異点付近にある場合は、ジョイントモーション関節動作の使用を推奨します。

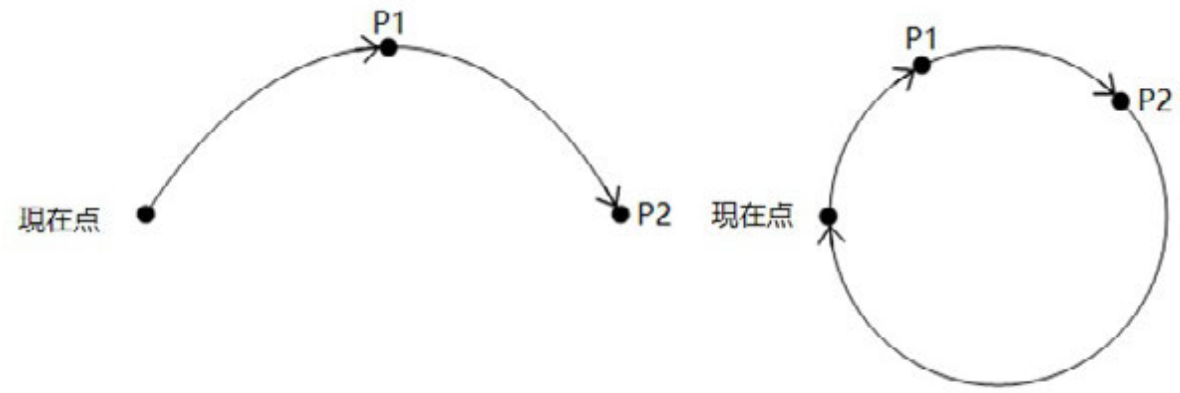
**直線運動**

ロボットアームは現在の姿勢と目標点の姿勢に基づいて動作軌跡を計画し、TCPの動作軌跡を直線に保ちながら、エンドエフェクタの姿勢が動作中に等速で変化するようにします。


動作軌跡が特異点を通過する場合、ロボットに直線運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

**円弧運動**

ロボットアームは現在位置、P1、P2の3点（これらは同一直線上にない）を基に、円弧または完全な円を決定します。動作中のロボットアームのエンドエフェクタの姿勢は、現在位置とP2の姿勢を補間して算出されます。P1の姿勢は計算に含まれません（つまり、動作中にロボットアームがP1に到達した際の姿勢は、ティーチング時の姿勢と異なる場合があります）。




動作軌跡が特異点を通過する場合、ロボットに円弧運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

## 座標系パラメータ

モーションブロックは、高度な設定にてポイントに対応するユーザー座標系とツール座標系を指定することができます。パラメータの優先順位は下記の通り：

1. 高度な設定にて座標系を指定した場合、指定した座標系を使用します。もしポイントはティーチングポイントであれば、ティーチングポイントのポーズ座標を指定した座標系の値に変換して使用します。
2. 未通过高级配置指定坐标系时，如果点位参数是示教点，使用示教点自带的坐标系索引。高度な設定で座標系を指定しなかった場合、ポイントパラメータがティーチングポイントであれば、ティーチングポイントに付属する座標系インデックスを使用します。
3. 高度な設定にて座標系を指定しない場合、もしポイントはジョイント変数またはポーズ変数であれば、**制御**ブロックグループの中にあるグローバル座標系を使用します（詳細は**ユーザー座標系設定**と**ツール座標系設定**のブロックをご参照ください、ブロック設定未使用の場合、デフォルト座標系は0となります）。

> **説明:**
> プロジェクトを実行すると、プロジェクトの実行前のジョグ画面で設定した座標系の値に関係なく、デフォルトのグローバル座標系が0に設定されます。

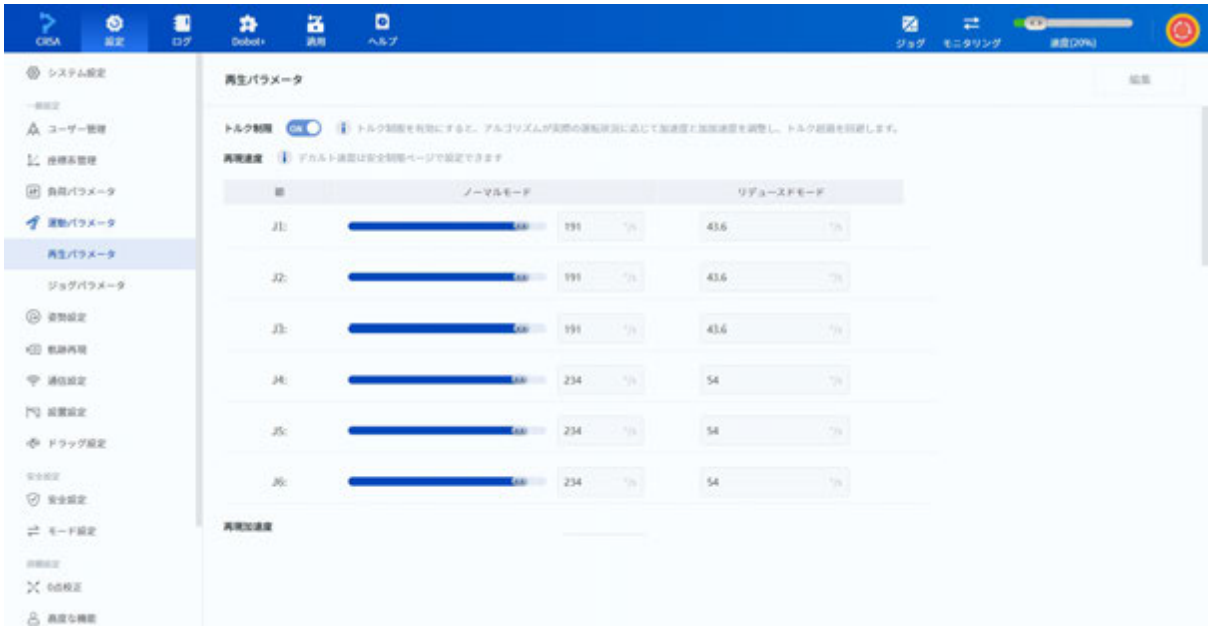
## 速度パラメータ

### 相対速度

モーションブロックは、高度な設定にてロボットが当該コマンドを実行する際の加速度（Accel）と速度比率（V）を指定することができます。

```
ロボット実際の運動速度 = 最大速度 x グローバル速度 x コマンド速度比率
ロボット実際の運動加速度 = 最大加速度 x コマンド速度比率
```

その最大速度/加速度は**再現パラメータ**によって制御され、DobotStudio Proの動作パラメータページで確認および変更できます。



グローバル速度は、DobotStudio Proの速度調整（上図右上隅）またはSpeedFactorコマンドで設定できます。

コマンド速度はモーションブロックの高度な設定で指定された速度を使用します。高度な設定で動作の加速度/速度の比率が指定されていない場合、デフォルトで対応する**関節/直線の速度/加速度の比率**ブロックで設定された値を使用します。ブロックが未設定の場合のデフォルト値はすべて100です。

**絶対速度**

直線及び円弧のモーションブロックの高度な設定では、モーションコマンドを実行する際のロボットの絶対速度（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ですが、**再現パラメータ**の最大速度の制限を受けます（ロボットがリデュースドモードに入っている場合、減速後の最大速度の制限を受けます）。つまり、絶対速度が再現パラメータ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に設定した場合、再現パラメータの最大速度2000より小さく、ロボットは目標速度1000mm/sとして運動します。この目標速度は、現時点のグローバル速度とは関係ありません。ただし、ロボットがリデュースドモード（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を目標速度として運動します。

絶対速度と速度比率は同時に設定することはできません。

## スムーズな遷移

特定の状況では、ロボットアームが目標地点に到達する前に、複数の遷移点を通過する必要があります。これらの遷移点は、通常、ロボットの動作経路が障害物を避けるために設けられたものであり、ロボットアームが正確に到達する必要はありません。スムーズトランジションパラメータを設定することで、ロボットアームは中継点に到達する前に曲がり始め、曲がる際の速度と軌跡がより滑らかになります。

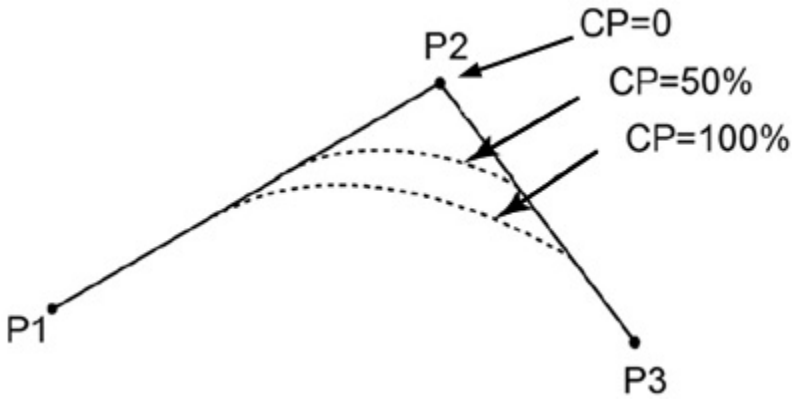
ユーザーはモーションコマンドの高度な設定を通じて、ロボットアームがこのモーションコマンドから次の動作指令へ移行する際のスムーズトランジション率（CP）またはスムーズトランジションの半径（R）を指定することができます。

**説明:**

- 関節運動モードでは、スムーズトランジション半径（R）の設定は対応していません。
- 指定した複数の通過点が異なるツール座標系に基づいている場合、スムーズな遷移は不可能になります。

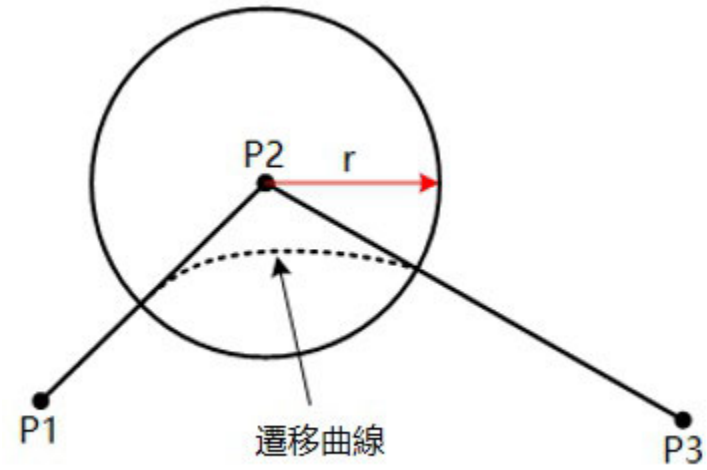
**CP**

スムーズトランジション率を設定すると、システムは自動的にトランジションカーブのラジアンを計算し、下図のように、CP値が大きいほど、曲線は滑らかになります。CP遷移曲線は運動速度/加速度の影響を受けます。ポイント位置とCP値が同じであっても、運動速度/加速度が異なると遷移曲線のラジアンは異なります。

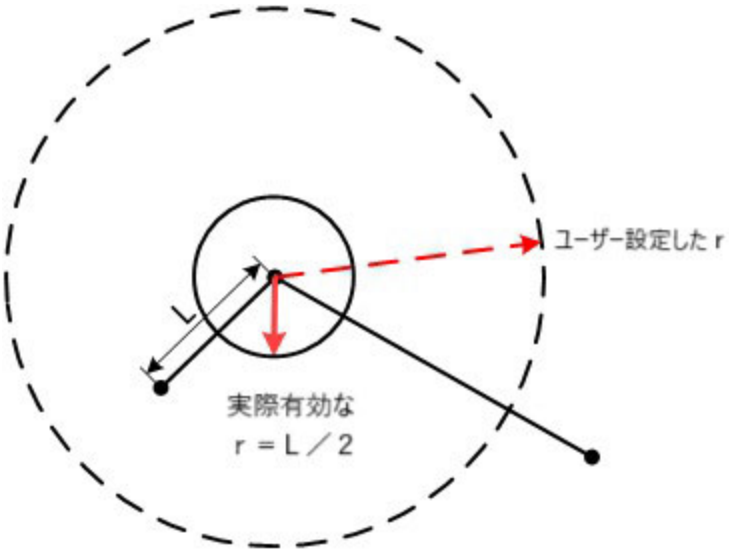



**R**

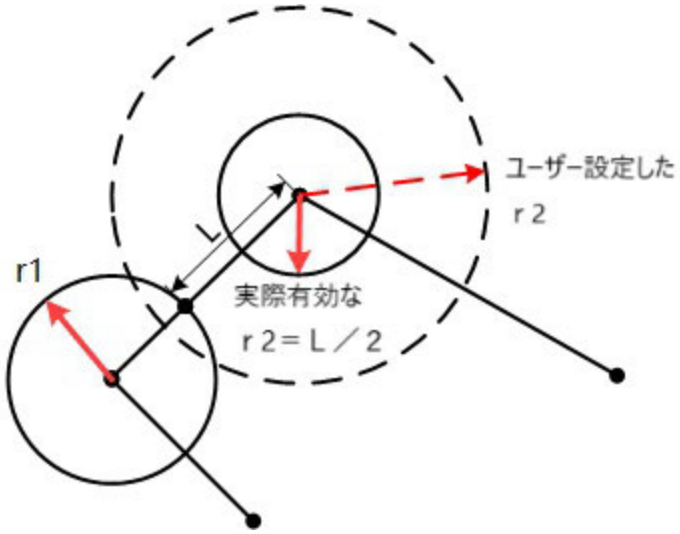
スムーズトランジション半径を設定する際、システムは遷移点を円の中心として取り、指定された半径に基づいて遷移曲線を計算します。R遷移曲線は運動速度/加速度の影響を受けず、ポイント位置と遷移半径のみで決定されます。




ユーザーが設定した遷移半径が大きすぎる場合（始点/終点と遷移点間の距離を超える）、システムは自動的に始点/終点間の短い距離（下図のL）の半分を遷移半径として遷移曲線を計算します。




連続する2つの遷移半径（下図のr1とr2）が重ねる場合、システムは1番目の運動セグメントの遷移が完了した後の点を2番目の運動セグメントの開始点として、遷移半径が大きすぎる場合の考え方として、実際の有効な2番目のrを計算します。




**デフォルト値**

高度な設定でスムーズトランジション率または半径が指定されていない場合、デフォルトで**スムーズな移行率** ブロックで設定された値が使用されます。ブロックの設定が使用されない場合、デフォルト値は0になります。

> **説明:**
> スムーズトランジションでは、ロボットの運動が中間点を通過しないため、スムーズトランジションを設定すると、2つのモーションコマンド間のIO信号出力や機能設定（例えば、セーフティスキンON/OFF）コマンドがトランジション処理中に実行されます。
> ロボットが正確に中間点に到達したときにコマンドを実行できるようにしたい場合、前のコマンドのスムーズトランジションのパラメーターを0に設定してください。

## 停止条件

一部の動作ブロックは、高度な設定で停止条件を指定することができます。ロボットがモーションコマンドを実行中に指定された停止条件を満たすと、現在の動作を終了し、次のコマンドを直接実行します。

- サポートされる判定条件には、IO、変数、およびLua文法に準拠した任意の式が含まれます。
- 最大で3つの判定条件を設定可能です。複数の条件は、キーワード**かつ**（すべての条件が満たされた場合に成立）/ **または**（いずれか1つの条件が満たされた場合に成立）で接続します。ただし、**かつ**と**または**を混在させることはできません。


**説明:**

- 停止条件を指定すると、スムーズトランジションパラメータrが無効になります。
- 停止条件を指定した後、スムーズトランジションパラメータcpが有効ですが、トランジション部分（曲線部分）が停止条件の有効範囲に入りません。
- 停止条件付きのコマンドの前のコマンドにスムーズトランジションを設定した場合、停止条件は運動セグメントの遷移が完了した後でトリガされます。

# 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、プログラムが実行を開始するための識別子として使用され、イベントブロックの下に接続されているブロックのみが実行されます。

## 運転を開始する


**説明:** プログラムのメインスレッドの識別子です。新しいプロジェクトを作成すると、プログラミングエリアには運転を開始するブロックが表示されます。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を作成してください。

**使用制限:** 一つのプロジェクトには一つの”運転を開始する”ブロックのみを使用することができます。

## サブスレッド起動


**説明:** プログラムのサブスレッドの識別子です。サブスレッドはメインスレッドと同時に実行されますが、サブスレッドはロボット制御コマンドを使用できず、変数演算またはIO制御のみを実行することができます。プロジェクトロジック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四つまでの”サブスレッド起動”ブロックを使用することができます。

**例:**

下記の例では、プロジェクトを実行した後、メインスレッドとサブスレッドが同時に実行を開始します。メインスレッドのモーションコマンドとサブスレッドのIOコマンドは相互に干渉せず、実行されます。それぞれの行列に従って実行されます。


運転を開始する
ずっと
MovL から 点 P1
MovL から 点 P2
サブスレッド起動
ずっと
もし デジタル入力 コントローラー DI_1 for オン なら
デジタル出力 コントローラー DO_1 オン を設定する

# 制御ブロックグループ

制御ブロックグループは、プログラムの実行パスを制御するために使用されます。

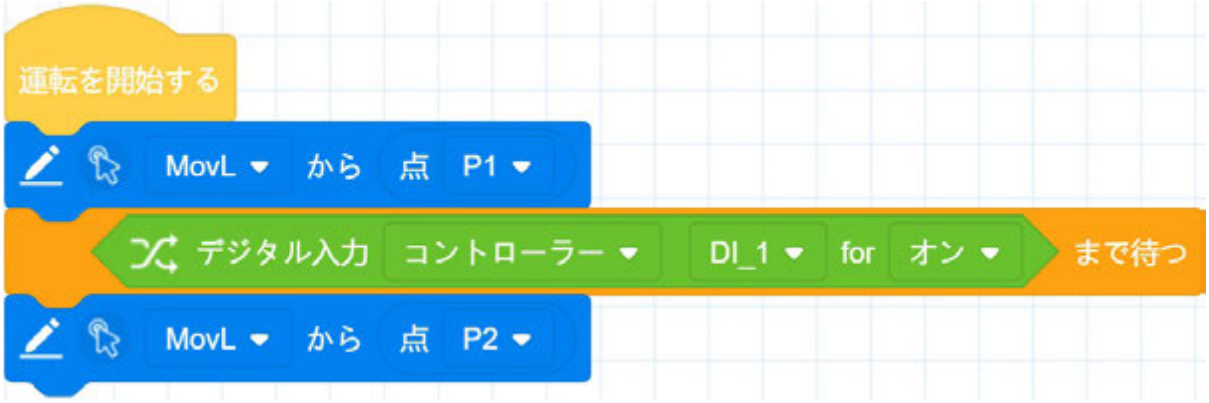
## 条件が満たされるまで待機


**説明:** プログラムを一時停止し、パラメータが `true` になるまで待機してから実行を続行します。

**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:**

ロボットアームがP1に移動し、DI1がオンになるまで待機してからP2に移動します。




## n回繰り返し


**説明:** 他のブロックをこのブロックの中に入れ、入れられたブロックのコマンドが指定された回数だけ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。

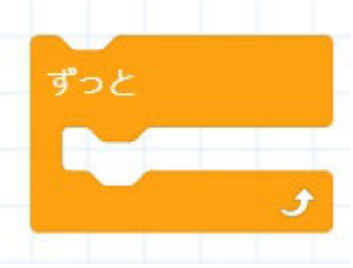
**例1:**

ロボットアームが2つのモーションコマンドを10回繰り返し実行します。


**例2:**

ロボットは2つのモーションコマンドをn回繰り返し、nは変数countの値となります。

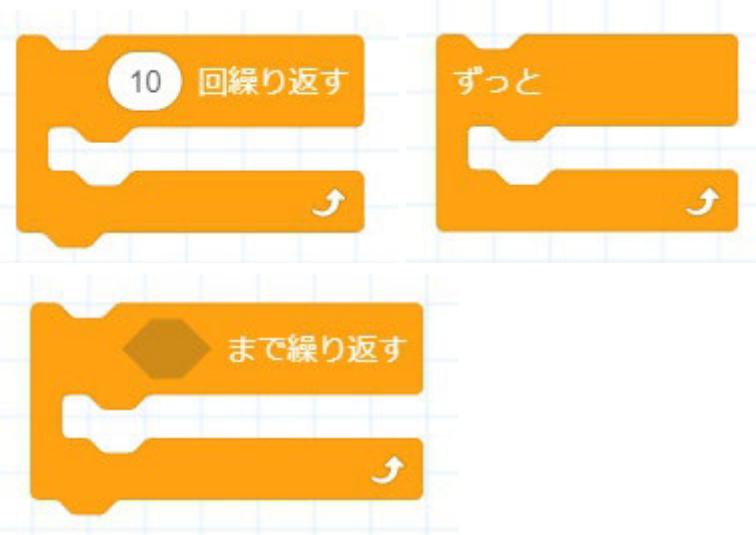

## 継続的に繰り返し




**説明:** 他のブロックをこのブロックの中に入れ、入れられたブロックのコマンドが**終了ブロック**に出会うまで繰り返し実行されます。このブロックの下に他のブロックを接続することはできません。

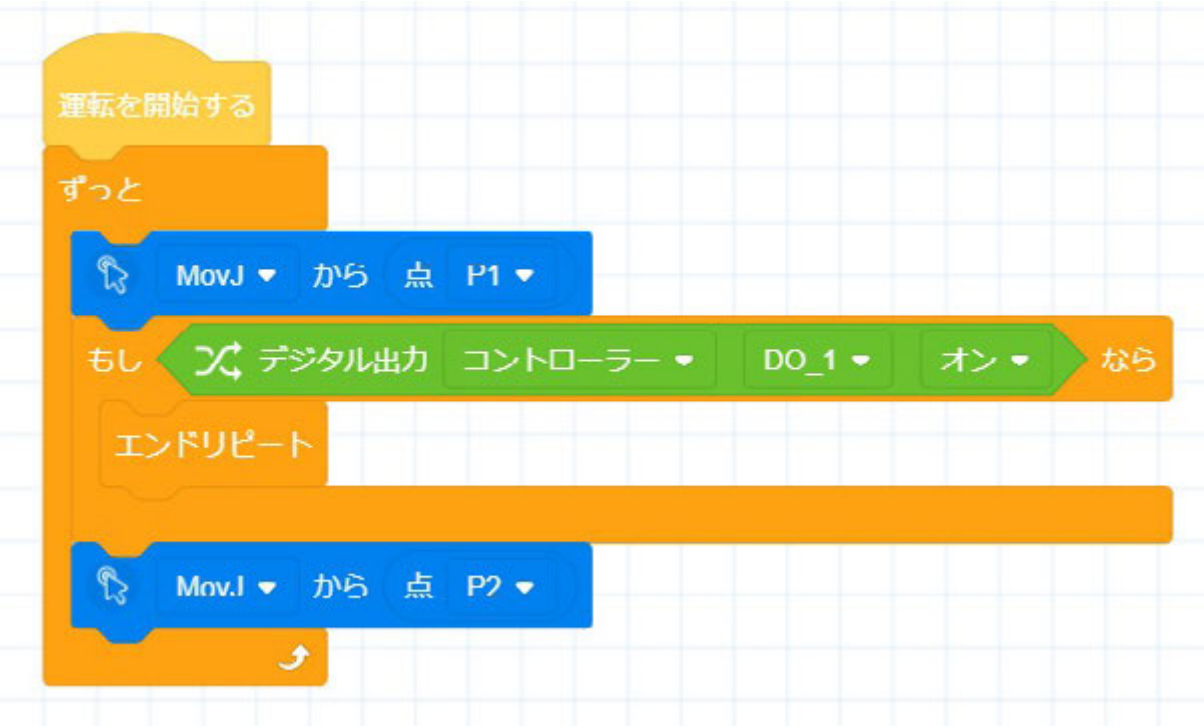
## 繰り返しを終了


**説明:** 以下の繰り返し実行系のブロックにネストして使用します。この基本ブロックに到達すると、繰り返しが直ちに終了し、繰り返しブロックの後に続くコマンドを実行します。




**例:**

P1とP2の往復運動をずっと繰り返す実行中に、ロボットがP1点に移動した後、DI1がONであれば、すぐに繰り返しを終了させます。




## 条件が満たされた場合に実行


**説明:** パラメータが `true` の場合、ネストされたブロック指令を実行します。パラメータが `false` の場合は、次のブロック指令に直接移動します。

**パラメータ:** 他の六角形ブロック（戻り値がブール値、すなわち `true` または `false` ）をパラメータとして使用します。

## 条件の成否によってそれぞれ実行


**説明:** パラメータが `true` の場合、「それ以外」の前にネストされたブロック指令を実行します。パラメータが `false` の場合、「それ以外」の後にネストされたブロック指令を実行します。

**パラメータ:** 他の六角形ブロック（戻り値がブール値、すなわち `true` または `false` ）をパラメータとして使用します。

**例:**

DI1がオンの場合、ロボットアームがP1点に移動します。それ以外の場合、P2点に移動します。


## 条件が満たされるまで繰り返し実行


**説明:** パラメータが `true` になるまで、ネストされたブロック指令を繰り返し実行します。

**パラメータ:** 他の六角形ブロック（戻り値がブール値、すなわち `true` または `false` ）をパラメータとして使用します。

## ラベルを設定


**説明:** ラベルを設定し、ラベルへジャンプするブロックにて当該ラベルへジャンプし実行します。

**パラメータ:** ラベル名称、ラベル名称の先頭は英字にする必要があり、スペースなどの特殊文字は使用できません。

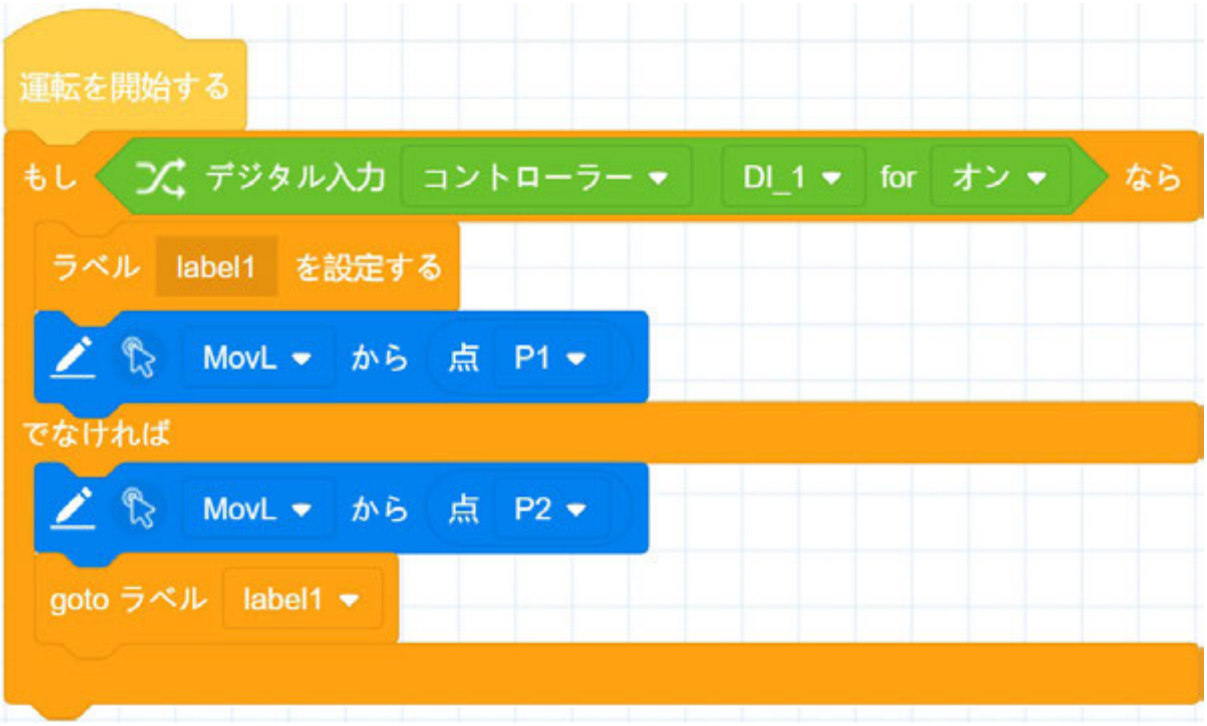
## ラベルにジャンプ

**説明:** プログラムは当該ブロックを実行した後、指定されたラベルへ直接ジャンプし、指定されたラベルの後のブロックコマンドを実行します。

**パラメータ:** 設定済みのラベル名称。

**例:**

もしDI1がオンなら、ロボットは直接P1に移動します、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。




**説明:**
当該ブロックは、同じ行列内のラベルブロック（サブルーチンを含む）にのみジャンプすることができます。他の行列へジャンプすることができません。例えば、カスタムブロックからメインスレッドにジャンプすることはできません。

## コマンドの折りたたみ

**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。制御機能はなく、プログラムをより美しく、読みやすくするための機能です。ブロックの右下にある二重矢印をクリックすると、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:** 折りたたまれるブロックを説明する名前を設定します。間接的で直感的な名前を付けることを推奨します。

## 一時停止

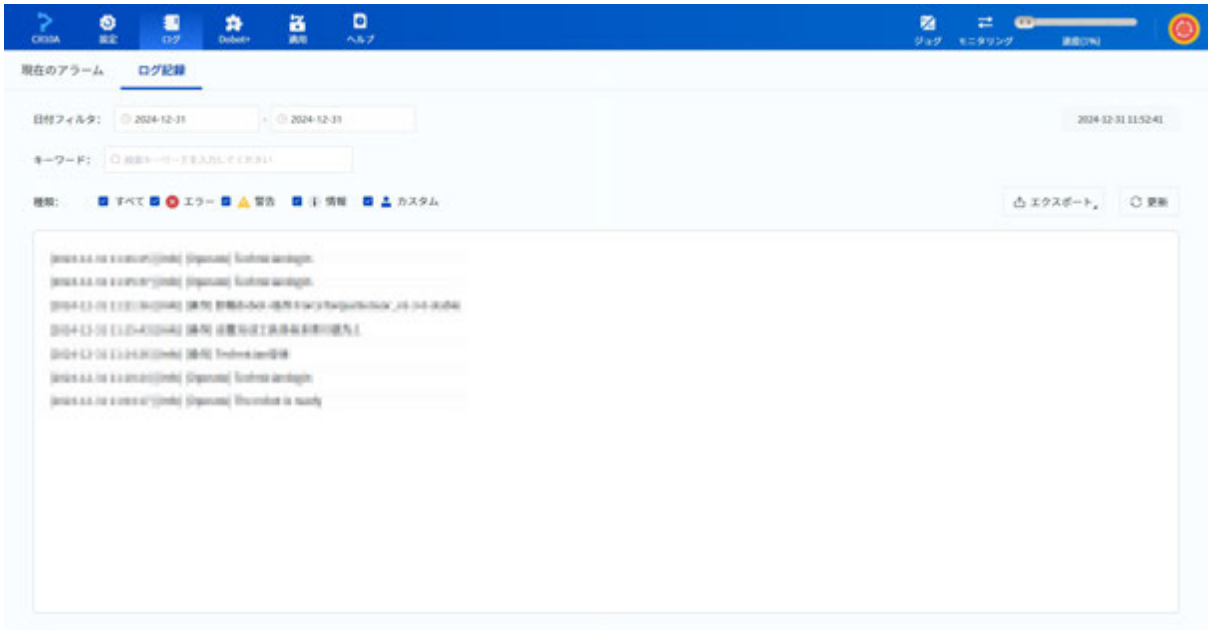

**説明:** プログラムは当該ブロックまで実行すると自動的に一時停止します。実行を継続するには、ソフトウェアまたはリモート制御を介して操作する必要があります。

## 停止プログラム


**説明:** プログラムがこのブロックに到達すると、自動的に停止して実行を終了します。

## カスタムログ


**説明:** カスタムのログ情報を出力します。ソフトウェアのログページにて表示またはエクスポートすることができます。




## 衝突検知機能を設定


**説明：**衝突検出機能を設定します。当該ブロックで設定した衝突検出レベルはプロジェクト実行中のみ有効になり、プロジェクトが停止すると元の値に戻されます。

**パラメータ：**衝突検出機能の感度を選択します。オフまたはレベル1～5から選択することができます。レベルの数字が大きいほど、衝突検出の感度が高くなります。

## 衝突時の後退距離を設定


**説明：**衝突を検知した後にロボットが元の軌道に戻るまでの距離を設定します。当該ブロックによって設定された値は、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元の値に戻されます。

**パラメータ：**衝突後退距離を設定します。値の範囲: [0,50]、単位：mm

## ユーザー座標系またはツール座標系を変更

**説明:** 指定されたユーザー座標系またはツール座標系を変更します。

**パラメータ:**

- ユーザー座標系或いはツール座標系のどちらを変更するかを指定します。
- 変更するユーザー座標系の番号を指定します。
- 変更するユーザー座標系のパラメータを指定します。
- 変更の有効範囲を指定する:
  - スクリプトのみ: 当該コマンドで変更した座標系は、現在のプロジェクト実行中にのみ有効になり、プロジェクトが停止すると元の値に戻されます。
  - グローバル保存: 当該コマンドで変更した座標系は、グローバルに保存され、変更された値はプロジェクトの停止後も保持されます。

**例:**


## ユーザー座標系を計算および更新


**説明:** 指定されたユーザー座標系を計算して更新します。この更新は、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元の値に戻されます。

**パラメータ:**

- 計算の基準として使用されるユーザー座標系の番号を指定します。ユーザー座標系0の初期値はベース座標系です。
- 計算の方向を指定します。
  - 左乗算: 前のパラメータで指定された座標系がベース座標系に沿ってオフセットされることを示します。
  - 右乗算: 前のパラメータで指定された座標系が現在座標系に沿ってオフセットされることを示します。
- 指定された座標系のオフセット値。
- 上記のパラメータを通じて新しいユーザー座標系を計算します。計算結果をどのユーザー座標系に更新するかを指定する必要があります。

**例:**

- ユーザー座標系**1の左乗算**のオフセット値: X **10** Y **10** Z **10** RX **10** RY **10** RZ **10**を計算し、結果を1に更新します。

上記のコマンドは、初期姿勢がユーザー座標系1と同じであり、ベース座標系を計算基準として{x=10, y=10, z=10}を平行移動し、{rx=10, ry=10, rz=10} を回転する新しい座標系をユーザー座標系1に更新することを示します。

- ユーザー座標系**1の右乗算**のオフセット値：X **10** Y **10** Z **10** RX **10** RY **10** RZ **10**を計算し、結果を1に更新します。

  上記のコマンドは、初期姿勢がユーザー座標系1と同じであり、ユーザー座標系1を計算基準として{x=10, y=10, z=10}を平行移動し、{rx=10, ry=10, rz=10} を回転する新しい座標系をユーザー座標系1に更新することを示します。

## ツール座標系を計算および更新


**説明：**指定されたツール座標系を計算して更新します。この更新は、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元の値に戻されます。

**パラメータ：**

- 計算の基準として使用され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。
- 計算の方向を指定します。
  - 左乗算：前のパラメータで指定された座標系がフランジ座標系に沿ってオフセットされることを示します。
  - 右乗算：前のパラメータで指定された座標系が現在座標系に沿ってオフセットされることを示します。
- 指定された座標系のオフセット値。
- 上記のパラメータを通じて新しいツール座標系を計算します。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**1の左乗算**のオフセット値：X **10** Y **10** Z **10** RX **10** RY **10** RZ **10**を計算し、結果を1に更新します。

  上記のコマンドは、初期姿勢がツール座標系1と同じであり、フランジ座標系を計算基準として{x=10, y=10, z=10}を平行移動し、{rx=10, ry=10, rz=10} を回転する新しい座標系をツール座標系1に更新することを示します。

- ツール座標系**1の右乗算**のオフセット値：X **10** Y **10** Z **10** RX **10** RY **10** RZ **10**を計算し、結果を1に更新します。

  上記のコマンドは、初期姿勢がツール座標系1と同じであり、ツール座標系1を計算基準として{x=10, y=10, z=10}を平行移動し、{rx=10, ry=10, rz=10} を回転する新しい座標系をツール座標系1に更新することを示します。

## ユーザー座標系を設定


**説明：** 現在のプロジェクトで使用するユーザー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。

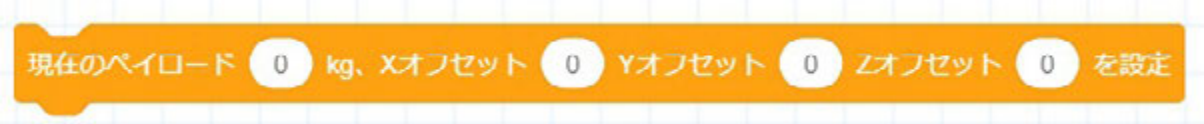
**パラメータ：** ユーザー座標系のインデックス。

## ツール座標系を設定


**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。

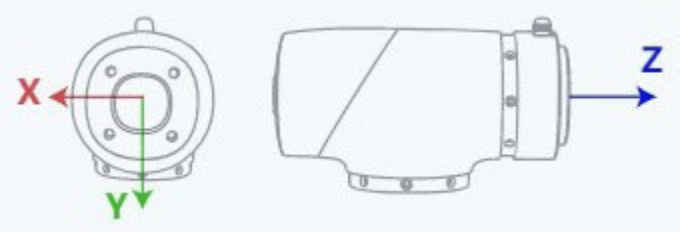
## 負荷パラメータを設定




**説明：** 末端負荷の重量とオフセット値を設定します。プロジェクトが停止すると元の値に戻されます。

**パラメータ：**

- 現在の負荷重量を入力します。単位：kg
- 現在のオフセット値を入力します。座標系の方向は下図にご参照ください。単位：mm。




## 指定時間待機


**説明:** 指定した時間を待機してから、次のコマンドを実行します。

**パラメータ:** コマンドの送信を遅延させる時間。待機時間の最大値は2147483647msで、設定したパラメータがこの最大値を超えると、コマンドは無効になります。

## セーフティウォールのOn/Offを設定


**説明:** 一つのセフティウォールを設定します。プロジェクトが停止すると元の値に戻されます。

**パラメータ:**

- セフティウォールの番号を選択します。現在選択したセフティウォールを削除した場合、疑問符が表示されます。
- セフティウォールの開閉状態を選択します。

## セーフティエリアのOn/Offを設定


**説明:** 一つの安全エリアを設定します。プロジェクトが停止すると、元の状態に戻ります。

**パラメータ:**

- 安全エリアのインデックスを選択します。選択した安全エリアが削除されると、インデックスは「? 」と表示されます。
- 安全エリアのオン/オフ状態を選択します。

## システム時間を取得


**説明:** 現在のシステム時間を取得します。

**戻り値:** システムの現在時間のUnixイムスタンプ。単位：ms。1970年1月1日の0:00GMTから現在時間までのミリ秒数に変換されます。時差を計算するために使用されます。現地の時間を取得するには、現地のタイムゾーンに従って取得されたグリニッジ標準時の変換を使用してください。

**例:**

取得した値が1686304295963の場合、北京時間へ変換すると、2023-06-09 17:51:35 (+963ms)となります。

取得した値が1686304421968の場合、北京時間へ変換すると、2023-06-09 17:53:41 (+968ms)となります。

複数回取得した値を減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


**説明:** プログラムは当該ブロックを実行した時、タイマーを開始します。以下の **タイマー結果を取得する**ブロックと組み合わせて使用する必要があります。

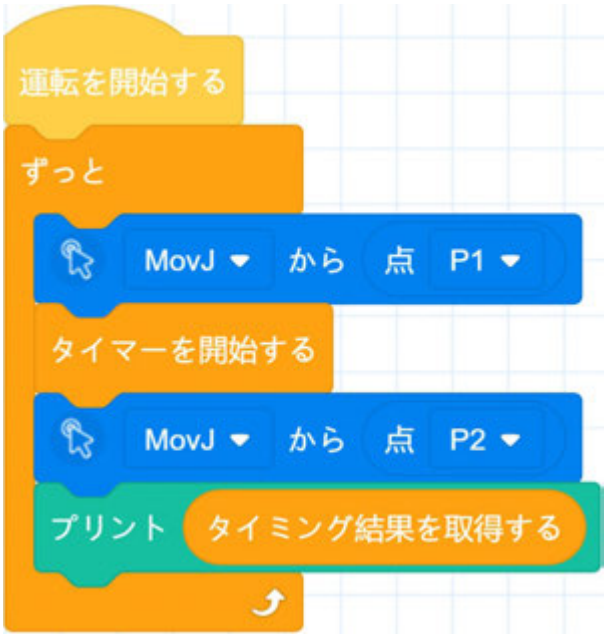
## タイマー結果を取得


**説明:** タイマーを終了します。時間差の値を反します。

**戻り値:** タイマー開始からタイマー終了までの時間差。単位：ms。最大時間は4294967295ms（約 49.7日）をカウントすることができます。最大時間を超えると、カウントは再び0から始まります。

**例:**

ロボットが関節運動でP1からP2へ移動する時間をプリントします。




## エンドツールモードを設定


**説明：** ロボット末端にAI1、AI2インターフェースと485インターフェースのリユース端子を備えた機種（CR、CRAシリーズ）の場合、末端ツールのモードを設定できます。デフォルトのモードは485モードです。

> **説明：**
> 末端ツールのモードを切り替えることを対応しない機種はこの設定は無効になります。

**パラメータ：**

- 末端ツールのモードを選択します。485モードとアナログ入力モードを対応します。485モードでは下記の二つのパラメータを設定する必要がありません。
- AI1アナログ入力モードを選択します。0～10V電圧入力モード、電流モード及び0～5V電圧入力モードを対応します。
- AI2アナログ入力モードを選択します。対応するモードはAI1と同じです。

## エンドツール485データフォーマットを設定


**説明:** エンドツールのRS485インターフェースの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。
- パリティビットの有無を選択します。
- ストップビットの長さを選択します。

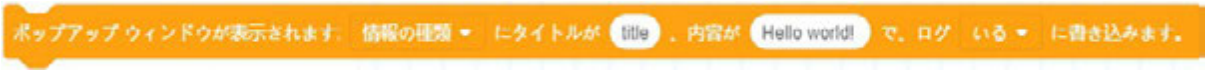
## エンドツール電源のオンオフを設定


**説明:** 末端ツールの電源状態を設定します。通常、末端電源の再投入に使用されます。例えば、エンドグリッパの初期化や再起動に使用されます。継続的にコマンドを呼び出す必要がある場合、間隔を最小4ms以上にすることを推奨します。

> **説明:**
> 末端電源をOffにした場合、末端DOも無効になります。

**パラメータ:** 電源状態のOn/Offを選択します。

## カスタムポップアップウィンドウ

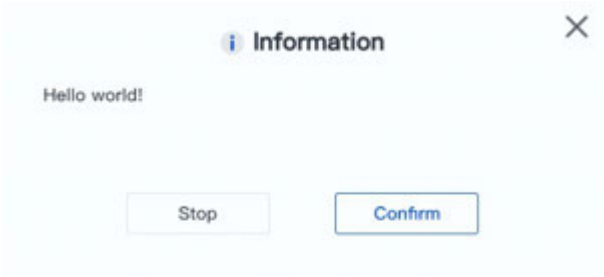



**説明:** スクリプトの実行中に、ユーザーがカスタマイズした情報を表示するウィンドウをポップアップします。DobotStudio Proでは、同時に1つのポップアップウィンドウのみが表示され、その内容は最新のものになります。

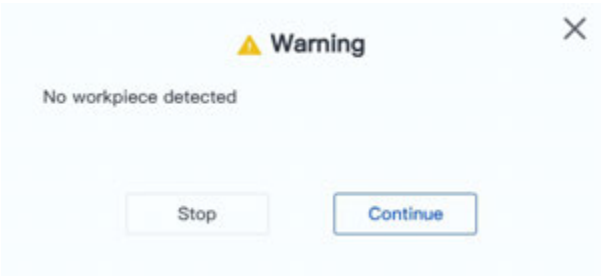
もしプロジェクト実行時にDobotStudio Proが起動していない場合（例えば、IOやModbusを通じてプロジェクトを開始した場合）、ポップアップは表示されません。しかし、アラームやエラータイプのポップアップコマンドは引き続きプロジェクトを一時停止させます。

**パラメータ:**

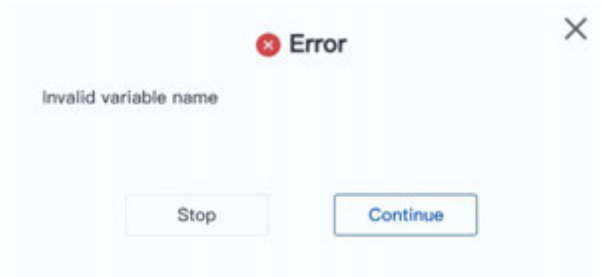
- メッセージタイプ。
  - 情報タイプ：このタイプのポップアップはプロジェクトの実行に影響を与えません。**停止**をクリックするとプロジェクトが停止し、**確認**をクリックするとポップアップが直接閉じられます。ポップアップが表示された後、ユーザーはIOやModbusを通じてプロジェクトを停止することもでき、プロジェクトが停止するとポップアップは自動的に消えます。




- アラームタイプ：このタイプのポップアップはプロジェクトを一時停止させます。**停止**ボタンをクリックするとプロジェクトが停止し、**続行**ボタンをクリックするとプロジェクトが再開します。ポップアップが表示された後、ユーザーはIOやModbusを通じてプロジェクトを停止または続行することもでき、プロジェクトが停止または続行されるとポップアップは自動的に消えます。




- エラータイプこのタイプのポップアップはプロジェクトを一時停止させます。**停止**ボタンをクリックするとプロジェクトが停止し、**続行**ボタンをクリックするとプロジェクトが再開します。ポップアップが表示された後、ユーザーはIOやModbusを通じてプロジェクトを停止または続行することもでき、プロジェクトが停止または続行されるとポップアップは自動的に消えます。




- 日志内容为“弹窗标题：弹窗内容”。表示するポップアップのタイトル。文字列のみサポートされ、長さは128文字以内です。

- 表示するメッセージ内容。文字列を含むさまざまな変数がサポートされています。文字列の場合、長さは128文字以内です。他の変数タイプの場合、文字列に変換され、変換後の文字数は512バイト以内でなければなりません。
- ポップアップ内容をログに記録するかどうか。
    - いいえ：ログに記録しません。
    - はい：対応するタイプのログに記録します（「情報」タイプのポップアップメッセージは「カスタム」タイプのログに対応します）。ログ内容は「ポップアップタイトル：ポップアップ内容」となります。


## コメント

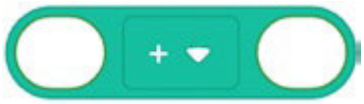

**説明：** コメントを追加します。コメントはプロジェクトの実行に影響を与えず、主にプロジェクトの閲覧者がロジックを理解するのを助けるために使用されます。

**パラメータ：** コメントの内容。

# 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**：パラメータで四則演算を行います。

**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します。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することができます。
- 中間に四則演算子を選択します。

**戻り値**：演算結果の数値。

## 比較演算


**説明**：パラメータで比較演算を行います。

**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します。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することができます。
- 中間に比較演算子を選択します。

**戻り値**：比較結果がtrueの場合はtrueを返し、falseの場合はfalseを返します。

## and演算


**説明**：パラメータでand演算を行います。

**パラメータ**：両側にand演算の変数を記入します。他の菱形ブロックを使用します。

**戻り値**：両方のパラメータがtrueの場合はtrueを返し、どちらか一方がfalseの場合はfalseを返します。

## or演算


**説明**：パラメータでor演算を行います。

**パラメータ**：両側にor演算の変数を記入します。他の菱形ブロックを使用します。

**戻り値**：両方のどちらか一方がtrueの場合はtrueを返し、両方がfalseの場合はfalseを返します。

## not演算


**説明**：パラメータでnot演算を行います。

**パラメータ**：not演算の変数を記入します。他の菱形ブロックを使用します。

**戻り値**：パラメータがtrueの場合はfalseを返し、falseの場合はtrueを返します。

## 剰余演算


**説明**：パラメータでnot演算を行います。

**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します。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することができます。

**戻り値**：演算結果の数値。

## 四捨五入演算


**説明**：パラメータで四捨五入演算を行います。

**パラメータ**：演算にかかわる変数または定数を記入します。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することができます。

**戻り値**：演算結果の数値。

## 単一値演算

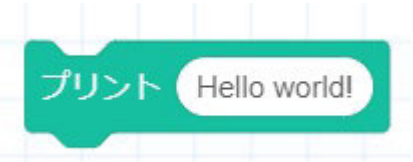

**説明**：パラメータで単一値演算を行います。三角関数（sin/cos/tan）の入力値と逆三角関数（asin/acos/atan）の戻り値は角度です。

**パラメータ**：

- 演算方法を選択します。
    - 絶対値
    - 切り捨て
    - 切り上げ
    - ルート（平方根）
    - sin
    - cos
    - tan
    - asin
    - acos
    - atan
    - ln
    - log
    - e^
    - 10^
- 演算にかかわる変数または定数を入力します。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することができます。

**戻り値**：演算結果の数値。

## プリント




**説明**：パラメータをコンソールに出力して表示します。主にデバッグに使用されます。

**パラメータ**：

コンソールに出力する変数または定数を入力します。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することができます。

# 文字列ブロックグループ

文字列ブロックグループには、文字列と配列の一般機能が含まれています。

## 文字列のn番目の文字を取得する


**説明**：文字列のn番目の文字を変数として取得します。

**パラメータ**：

- 1番目のパラメータは文字列を入力します。他の楕円形ブロックを使用するか直接記入することができます。
- 2番目のパラメータは文字列の何番目の文字を返すかを指定します。

**戻り値**：文字列の指定位置の文字。

## 文字列1に文字列2が含まれるか否かを判断する


**説明**：1番目のパラメータの文字列に2番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。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することができます。

**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、それ以外の場合はfalseを返します。

## 2つの文字列を結合する


**説明**：2つの文字列を1つの文字列に結合します。2番目の文字列は1番目の文字列の後につきます。

**パラメータ**：結合する2つの文字列。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することができます。

**戻り値**：結合後の文字列。

## 文字列または配列の長さを取得する


**説明**：指定された文字列または配列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ます。配列の長さとは、この配列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配列。戻り値が文字列または配列の楕円形ブロックを使用することができます。

**戻り値**：文字列または配列の長さの値。

## 2つの文字列を比較する


**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。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することができます。

**戻り値**：文字列1と文字列2が同じの場合は0を返し、文字列1が文字列2より小さい場合は-1を返し、文字列1が文字列2より大きい場合は1を返します。

## 配列を文字列に変換する


**説明**：指定された配列を文字列に変換します。文字列にあるそれぞれの配列の要素を指定の区切り文字で区切ります。例えば、配列は{1,2,3}、区切り文字は|の場合、変換後の文字列は”1|2|3”です。

**パラメータ**：

- 文字列に変換する配列。戻り値が配列の楕円形ブロックを使用します。
- 変換に使用される区切り文字。

**戻り値**：変換後の文字列。

## 文字列を配列に変換する


**説明:** 指定された文字列を配列に変換します。指定の区切り文字によって文字列を区切ります。例えば、文字列は“1|2|3”、区切り文字は|の場合、変換後の配列は{[1]=1,[2]=2,[3]=3}です。

**パラメータ:**

- 配列に変換する文字列。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することができます。
- 変換に使用される区切り文字。

**戻り値:** 変換後の配列。

## 指定された配列の要素を取得する


**説明:** 指定された配列の要素を取得します。インデックスとは、配列におけるる要素の位置を表します。例えば、配列{7,8,9}における8のインデックスは2です。

**パラメータ:**

- 対象配列。戻り値が配列の楕円形ブロックを使用します。
- 要素のインデックスを指定します。

**戻り値:** 配列における指定された要素の値。

## 指定された配列の複数の要素を取得する


**説明:** 指定された配列の複数の要素を取得します。開始インデックスから終了インデックスまでの範囲内で、ステップ値によって要素を取得します。

**パラメータ:**

- 対象配列。戻り値が配列の楕円形ブロックを使用します。
- 開始インデックスと終了インデックスを通じて、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ます。1はすべてを取得します。2は要素を1つおきに取得します。以降同様です。

**戻り値:** 指定された要素で構成した新しい配列。

## 指定された配列の要素を設定する


**説明：** 指定された配列の要素の値を設定する。

**パラメータ：**

- 対象配列。戻り値が配列の楕円形ブロックを使用します。
- 要素のインデックスを指定します。
- 要素の値を指定します。

# カスタムブロックグループ

カスタムブロックグループは、カスタムブロックの作成と管理、及びグローバル変数の呼び出しに使用されます。

## グローバル変数を呼び出す


**説明:** 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。現在選択されているグローバル変数は、削除されると疑問符として表示されます。

**戻り値:** グローバル変数の値。

## グローバル変数を設定する


**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。現在選択されているグローバル変数は、削除されると疑問符として表示されます。
- 変更後の値。他の楕円形ブロックを使用するか直接記入することができます。

## カスタム変数を新規作成する


クリックするとカスタム変数を新規作成することができます。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。

## カスタム数値変数の値を設定する


**説明**: 指定された数値変数を設定します。需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。他の楕円形ブロックを使用するか直接記入することができます。

## カスタム数値変数の値を増減する


**説明**: 指定した数値の変数を指定した値分で増加します。

**パラメータ**:

- 変更する変数の名称を選択します。
- 増加する値。他の楕円形ブロックを使用するか直接記入することができます。マイナスに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。

## カスタム文字列変数の値を設定する


**説明**：指定された文字列変数を設定します。

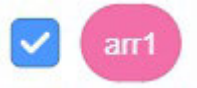
**パラメータ**：

- 変更する変数の名称を選択します。
- 変更後の値。文字列を直接記入します。

## 配列を作成する


クリックするとカスタム配列を新規作成することができます。配列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つの配列を作成した後、ブロックリストに下記の配列関連のブロックが現れます。

## カスタム配列




**説明**：新規作成したカスタム配列で、デフォルトは空の配列です。値を割り当てた後に使用することを推奨します。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、配列の名称を変更したり、配列を削除したりすることができます。その他の配列ブロック中の配列選択ドロップダウンリストにより、現在選択した配列の名称を変更したり、この配列を削除したりすることができます。配列ブロック前のチェックボックス機能は現在無効です。無視してください。

**戻り値**：配列の値。

## 変換を配列に追加する


**説明**：変数を指定された配列に追加します。新たに追加する変数は、配列の最後の1つ項目になります。

**パラメータ**：

- 追加する変数。他の楕円形ブロックを使用するか直接記入することができます。

- 変更する配列を選択します。

## 指定された配列の項目を削除する


**説明**: 指定された配列の項目を削除します。

**パラメータ**:

- 変更する配列を選択します。
- 削除する項目の番号。戻り値が数値の楕円形ブロックを使用するか直接記入することができます。

## 配列のすべての項目を削除する


**説明**: 指定された配列のすべての項目を削除します。

**パラメータ**: 変更する配列を選択します。

## 配列に項目を挿入する


**説明**: 配列に指定された位置に変数を挿入します。

**パラメータ**:

- 変更する配列を選択します。
- 挿入する位置。戻り値が数値の楕円形ブロックを使用するか直接記入することができます。
- 追加する変数。他の楕円形ブロックを使用するか直接記入することができます。

## 指定された配列の項目を置換する

**説明**：指定された配列の項目を指定された変数に置換します。

**パラメータ**：

- 変更する配列を選択します。
- 置換する項目の番号。戻り値が数値の楕円形ブロックを使用するか直接記入することができます。
- 置換後の変数。他の楕円形ブロックを使用するか直接記入することができます。

## 指定された配列の項目を取得する

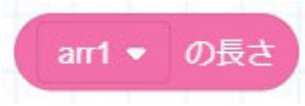

**説明**：指定された配列の項目の値を取得します。

**パラメータ**：

- 項目を取得する配列を選択します。
- 取得する項目の番号。戻り値が数値の楕円形ブロックを使用するか直接記入することができます。

**戻り値**：指定された項目の値。

## 配列の項目の総数を取得する




**説明**：配列の項目の総数を取得します。

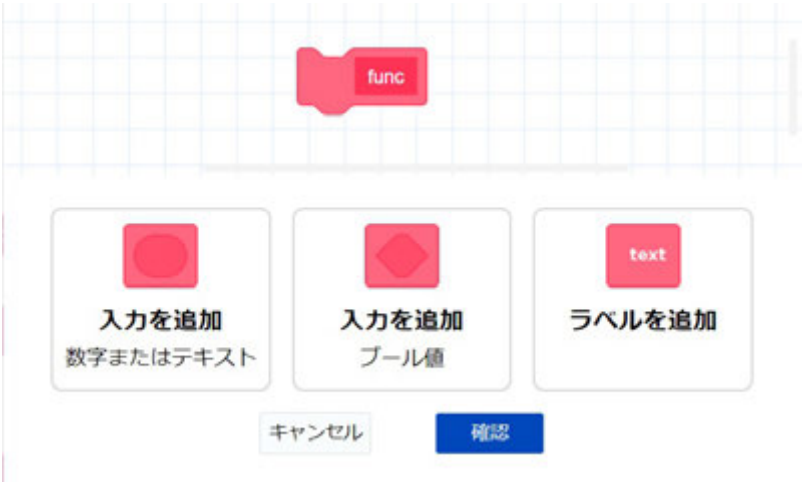
**パラメータ**：項目を取得する配列を選択します。

**戻り値**：指定された項目の総数。

## 関数を新規作成する


クリックすると1つの関数を新規作成することができます。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。

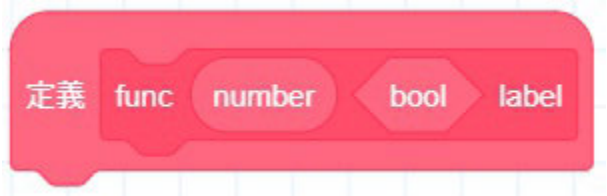
1. **関数の宣言**




この画面では、関数の名称、入力（パラメータ）のタイプ、数量と名称を定義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することができます。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。

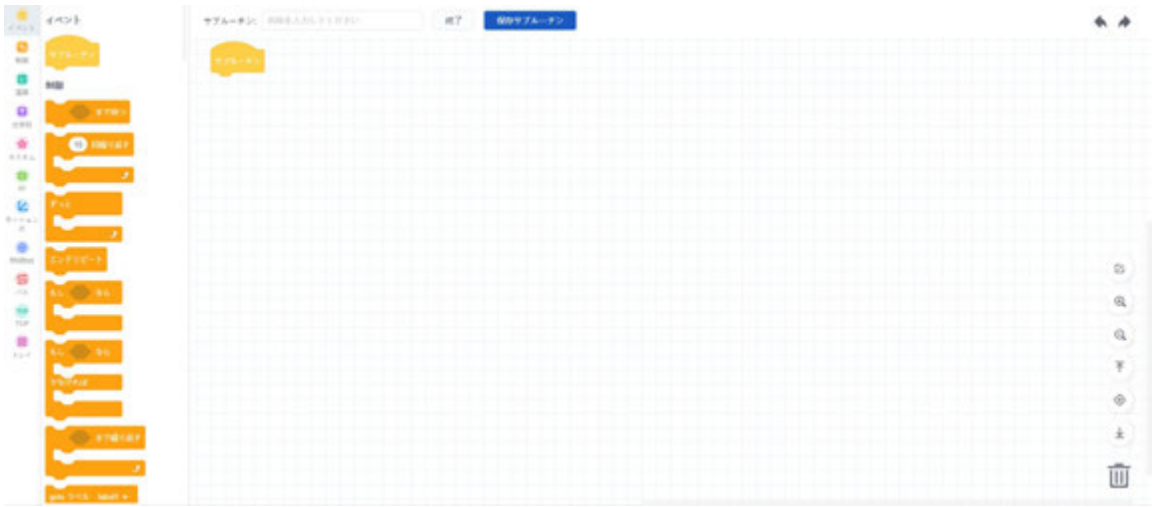
## サブルーチンを新規作成する


クリックすると、サブルーチンを新規作成することができます。

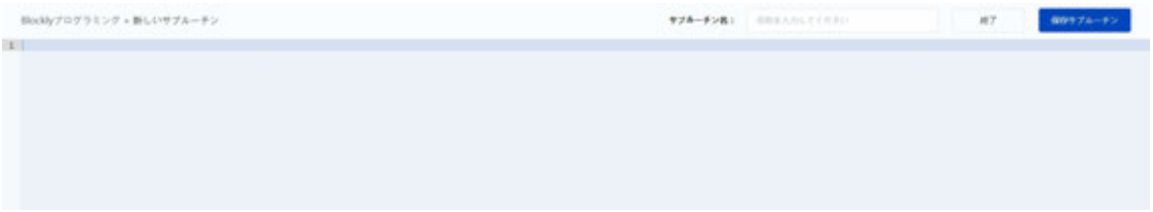
> **説明:**
> サブルーチンと関数の違いは、サブルーチンはカプセル化されておらず、本質的に再利用可能なコードブロックであることです。サブルーチンを呼び出すことの効果は、サブルーチン内のコードを対応する場所にコピーして貼り付けることと同じです。

Blocklyプログラミングは、入れ子されたサブルーチンを呼び出すことを対応します。サブルーチンは、最大2つの入れ子でBlocklyプログラミングとスクリプトプログラミングを対応します。新しいサブルーチンが正常に作成されると、対応するサブルーチンブロックがブロックリストに表示されます。


- Blocklyプログラミングを選択すると、画面がサブルーチンブロック プログラミング画面に変わります。サブルーチンの説明を設定したり、サブルーチンを作成したりすることができます。



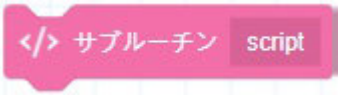
- スクリプトプログラミングを選択すると、サブルーチンスクリプトプログラミングウィンドウが表示されます。サブルーチンの説明を設定したり、サブルーチンを作成したりすることができます。



## サブルーチン

- Blocklyプログラミングサブルーチン


- スクリプトプログラミングサブルーチン




**説明:** サブルーチンブロックの説明は、サブルーチンの作成時にユーザーによって定義されます。保存された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）することで、サブルーチンを変更または削除することができます。

# 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ターミナルの入出力を管理するために使用されます。入出力ポートの値の範囲はロボットの対応する端子数により決まりますので、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。

## デジタル出力を設定する


**説明:** 指定された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。コントローラーと末端を含みます。
- 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）。

## デジタル出力状態を判断する


**説明:** 指定された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。コントローラーと末端を含みます。
- DO端子の番号を選択します。
- trueと判断する状態を選択します。

**戻り値:** 指定されたDOの現在の状態が条件を満たしている場合はtrueを返し、それ以外の場合はfalseを返します。

## 一連のデジタル出力を設定する


**説明:** 一連のDO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、クリックすると、DOのand状態を設定します。


- + あるいは - により、設定するDOの数を増減することができます。
- 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を選択します（オンまたはオフ）。

## 一連のデジタル出力を待つ


**説明：** 一連のDOの状態がすべて条件を満たした後、次のコマンドを実行します。

**パラメータ：**

- 待つ状態を選択します（オンまたはオフ）。
- タイムアウト待つ時間を設定します。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てから、クリックすると、待つDOを設定します。


- + あるいは - 待機するDOの数を増減することができます。
- 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。

## デジタル入力を待つ


**説明:** 指定されたDIが条件を満たすまで待機するか、タイムアウト時間を待ってから次のブロックコマンドを実行します。

**パラメータ:**

- DI端子の位置を選択します。コントローラーと末端を含みます。
- DI端子の番号を選択します。
- 待つ状態を選択します（オンまたはオフ）。
- タイムアウト待つ時間を設定します。0に設定すると、条件が満たされるまで待ちます。

## 一連のデジタル入力を待つ


**説明:** 一連のDIの状態がすべて条件を満たした後、次のコマンドを実行します。

**パラメータ:**

- 待つ状態を選択します（オンまたはオフ）。
- タイムアウト待つ時間を設定します。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てから、クリックすると、待つDIを設定し

ます。


- + あるいは - 待機するDIの数を増減することができます。
- 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**：指定されたアナログ出力の値を設定します。値（電圧/電流）の意味は、DobotStudio Proの**モニタリング > コントローラーAI/AO**画面で確認及び変更することができます。

**パラメータ**：

- アナログ出力端子の番号を選択します。
- 出力値。戻り値が数値の楕円形ブロックを使用するか直接記入することができます。

## 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。値（電圧/電流）の意味は、DobotStudio Proの**モニタリング > コントローラーAI/AO**画面で確認及び変更することができます。

**パラメータ**：アナログ出力端子の番号を選択します。

**戻り値**：AOの現在の電圧値または電流値。

## デジタル入力状態を判断する


**説明**：指定された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。コントローラーと末端を含みます。
- DI端子の番号を選択します。
- trueと判断する状態を選択します。

**戻り値**：指定されたDIの現在の状態が条件を満たしている場合は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**：指定されたアナログ入力の値を取得します。

コントローラーのアナログ入力値（電圧/電流）の意味は、DobotStudio Proの**モニタリング > コントローラーAI/AO**画面で確認及び変更することができます。

末端アナログ入力を取得するには、先に末端ツールのモードを設定するブロックで末端ツールモードをアナログ入力モードに設定する必要があります。

**パラメータ**：

- アナログ入力端子の位置を選択します。コントローラーと末端を含みます。
- アナログ入力端子の番号を選択します。

**戻り値**：指定されたアナログ入力の値。

# モーションブロックグループ

モーションブロックグループは、ロボットのモーション制御やモーション関連の設定を行うために使用されます。

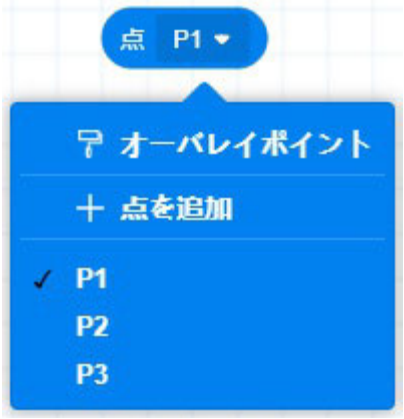
ポイントパラメータは、プロジェクトの**ポイントリスト** に追加した後、選択することができます。モーション制御ブロックは、デフォルトの変数ブロックをドラッグアウトし、他の戻り値がポイントの楕円形ブロックで入れ替えることができます。

## 目標点への移動


**説明:** ロボットアームを現在位置から指定されたポイントまで移動させます。プログラミングエリアにブロックをドラッグすると、高度な設定が可能になります。

**パラメータ:**

- 移動方式を選択します。関節移動と直線移動がサポートされています。
- 目標ポイント。デフォルトのポイント変数ブロックは以下の機能をサポートします:




- 保存されたポイントリストからポイントを選択します。
- ロボットアームの現在の姿勢を新しいポイントとして追加し、自動的に選択します。
- ロボットアームの現在の姿勢で現在選択されているポイントを上書きします。

**高度な設定**

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共通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。
- ユーザー座標系：点パラメータのユーザー座標系インデックス。
- ツール座標系：点パラメータのツール座標系インデックス。
- プロセスI/O設定：


**プロセスIOの設定と停止条件を同時に選択することはできません。**

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。

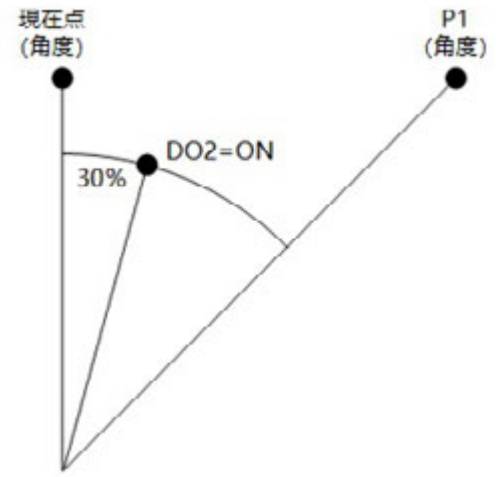
移動方式が関節移動の場合、距離は各関節角の合成角ベクトルを表し、計算は複雑になります。そのため、百分率モードを使用することを推奨し、効果はより直感的です。

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。

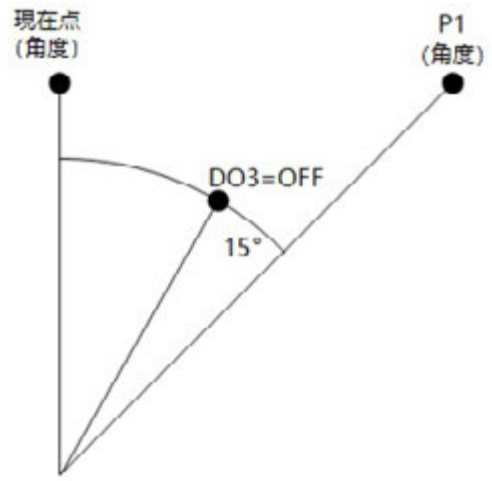
**例1**：P1まで関節移動、DOインデックス：DO_02、DO状態：ON、トリガモード：百分率、距離：30%

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:


**プロセスIOの設定と停止条件を同時に選択することはできません。**

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

**例1**: 条件を1行だけ設定した場合「DI1==ONの場合」、この動作命令の実行中にDI1がONであることを検知すると、現在の動作をスキップします。

**例2**: 条件を2行設定した場合「DI1==ONかつDI2~=ONの場合」、この動作命令の実行中にDI1がONかつDI2がONではないことを検知すると、現在の動作をスキップします。

**例3**: 条件を2行設定した場合「DI1==ON、またはグローバル変数var<=10の場合」、この動作命令の実行中にDI1がONであるか、グローバル変数varが10以下であることを検知すると、現在の動作をスキップします。

## 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、クリックして高度な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。
- ロボットがユーザー座標系か、またはツール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は空間偏移量を示します。単位：mm。rx，ry，rzは角度偏移量を示します。単位：度。

**高度な設定**

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共通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。
- ユーザー座標系：オフセット時に参照したユーザー座標系のインデックス。
- ツール座標系：オフセット時に参照したツール座標系のインデックス。
- 停止条件:


動作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動作を終了し、次のコマンドを直接実行します。

## 座標系オフセット後の点を取得


**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。
- 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。ユーザー座標系またはツール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ポイント位置。

## 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。単位：度。

**高度な設定**


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共通説明を参照してください。

- 速度比率 (V): 動作速度の比率、範囲は1～100。
- 加速度 (Accel): 動作加速度の比率、範囲は1～100
- スムーズトランジション (CP): スムーズな移行の比率、範囲は0～100。
- ユーザー座標系 / ツール座標系: このブロックでは、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を取得


**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。

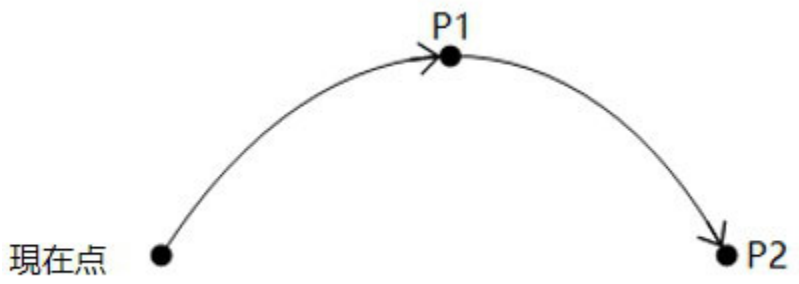
**パラメータ:**

- 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ポイント位置。

## 円弧動作を実行


**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。
- 終了点は目標点です。

**高度な設定**

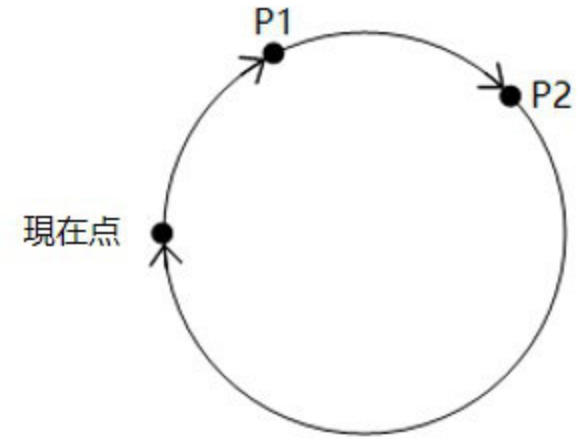

以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は共通説明を参照してください。

- 速度比率 (V): 動作速度の比率、範囲は1～100。
- 絶対速度 (Speed): 動作の絶対速度値。Vと相互排他。
- 加速度 (Accel): 動作加速度の比率、範囲は1～100。
- スムーズトランジション (CP): スムーズな移行の比率、範囲は0～100。
- トランジション半径 (R): トランジション曲線の半径。CPと相互排他。
- ユーザー座標系: ポイントパラメータのユーザー座標系インデックス。
- ツール座標系* ポイントパラメータのツール座標系インデックス。
- 停止条件:

動作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動作を終了し、次のコマンドを直接実行します。

## 円周動作を実行


**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、クリックして高度な設定を行うことができます。




動作中のロボットアームのエンドエフェクタの姿勢は、現在のポイントとP2ポイントの姿勢を補間して算出されます。P1ポイントの姿勢は計算に含まれません（つまり、動作中にロボットアームがP1ポイントに到達した際の姿勢は、ティーチング時の姿勢と異なる可能性があります）。

**パラメータ：**

- 中間点: 円全体を決定するための位置決めポイント1。
- 終了点: 円全体を決定するための位置決めポイント2。
- 円運動を行う周回数を入植します。範囲：1～999。

**高度な設定**


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共通説明を参照してください。

- 速度比率 (V): 動作速度の比率、範囲は1～100。
- 絶対速度 (Speed): 動作の絶対速度値。Vと相互排他。
- 加速度 (Accel): 動作加速度の比率、範囲は1～100。
- スムーズトランジション (CP): スムーズな移行の比率、範囲は0～100。
- トランジション半径 (R): トランジション曲線の半径。CPと相互排他。
- ユーザー座標系: ポイントパラメータのユーザー座標系インデックス。
- ツール座標系* ポイントパラメータのツール座標系インデックス。
- 停止条件:


  動作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動作を終了し、次のコマンドを直接実行します。

## 軌跡再現


**説明：** ロボットアームは軌跡の始点まで移動した後、軌跡を再現します。再現する軌跡ファイルは軌跡再現プロセスで記録する必要があります。プログラムエリアにブロックをドラッグすると、高度な設定が可能になります。

**パラメータ：**

- 軌跡の始点まで移動する際の移動方式を選択します。関節移動と直線移動がサポートされています。
- 再現する軌跡ファイルを選択します。選択中の軌跡ファイルが削除されると、「?」と表示されます。
- 再現時の動作速度を選択します：
  - 等速: ロボットアームはグローバル速度設定に基づき一定速度で軌跡を再現します。
  - 0.25倍速: 軌跡記録時の元の速度を0.25倍に等比縮小。ロボットアームの動作速度はグローバル速度設定の影響を受けません（以下同様）。
  - 0.5倍速
  - 1倍速
  - 2倍速

**高度な設定**


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。
- ユーザー座標系：軌跡点位置に対応するユーザー座標系インデックスを指定します。指定していないとき、軌跡ファイル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。
- 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。

## スムーズな遷移の比率を設定


**説明**：移動中のスムーズトランジション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。スムーズトランジションの詳細な説明は共通説明を参照してください。

**パラメータ**：スムーズな遷移の比率で、値範囲は0~100です。

## 関節速度の比率を設定


**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共通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。

## 関節加速度の比率を設定


**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共通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。

## 直線速度の比率を設定


**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共通説明を参照してください。

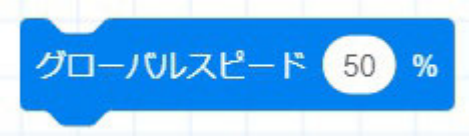
**パラメータ**：直線および円弧の速度比率で、値範囲は0~100です。

## 直線加速度の比率を設定


**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共通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。

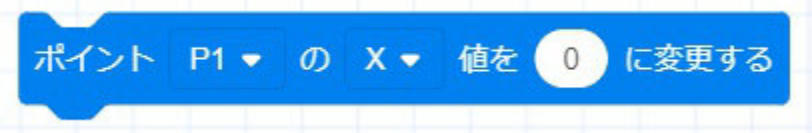
## グローバル速度の比率を設定




**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共通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。

## 指定点の座標値を変更




**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。

## 指定点の座標値を取得


**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。

## 動作の実現可能性をチェック


**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: 終点が肩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
- 30: 全パラメータの逆計算が解なし
- 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

## 指定点の指定座標次元の値を取得


**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。

## 指定点の指定関節の角度を取得


**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。

## 関節角度を正計算で位姿に変換


**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。
- ツール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} }`

## 位姿を逆計算で関節角度に変換

**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。
- ツール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。

## エンコーダの値を取得


**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ブロックグループ

Modbusブロックグループは、Modbus通信に関連する操作を実行するために使用されます。対応するDEMOを参照して、関連するコマンドの使用をすぐに体験することができます。

各種データに対応する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 | 単一書き込み | 連続書き込み |
|---|---|---|---|
| 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 |

## Modbusマスターステーションを作成


**説明:** コントローラーのイーサネットポートを使用してModbus TCPマスターステーションを作成し、スレーブと接続を確立します。最大で同時に15台のスレーブと接続することが可能です。

**パラメータ:**

- ModbusスレーブのIPアドレスを入力します。
- Modbusスレーブのポート番号を入力します。
- ModbusスレーブのIDを選択します。

ロボット内蔵のスレーブに接続する場合、IPアドレスはロボットのIP（デフォルトは192.168.5.1、変更可能）を設定し、ポート番号は502（map1）または1502（map2）に設定します。詳細は付録A Modbusレジスタ定義を参照してください。

サードパーティ製のスレーブに接続する場合、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の説明を参照してください。

## エンドRS485ベースのModbusマスターステーションを作成


**説明:** エンドエフェクタのRS485インターフェースを使用してModbus TCPマスターステーションを作成し、スレーブデバイスと接続を確立します。最大で同時に15台のデバイスと接続することが可能です。

**パラメータ:**

- ModbusスレーブのIPアドレスを入力します。
- RS485インターフェースのボーレートを入力します。
- ModbusスレーブのIDを選択します。
- パリティビットの有無を選択します。
- ストップビットの長さを選択します。

ロボット内蔵のスレーブに接続する場合、IPアドレスはロボットのIP（デフォルトは192.168.5.10、変更可能）を設定し、ポート番号は60000（設定および変更不可）に設定します。

サードパーティ製のスレーブに接続する場合、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の説明を参照してください。

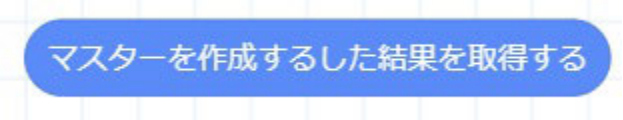
## RS485ベースのModbusマスターステーションを作成


**説明:** コントローラーのRS485インターフェースを使用してModbus RTUマスターステーションを作成し、スレーブデバイスと接続を確立します。最大で同時に15台のデバイスと接続することが可能です。

**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スレーブのIDを選択します。
- パリティビットの有無を選択します。
- データビットの長さを入力します。現在のバージョンでは8のみサポートしています。
- ストップビットの長さを選択します。

## マスターステーション作成結果を取得




**説明:** Modbusマスターを作成した結果を取得します。

**戻り値:**

- 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既に15個あり、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してください。
- 3：スレーブへの接続に失敗しました。スレーブが正常に作成されているか、ネットワーク

が正常であるかなどを確認してください。

## 入力レジスタを待機


**説明:** 指定された入力レジスタのアドレスのデータが条件を満たすと、次のコマンドを実行します。

**パラメータ:**

- 入力レジスタの先頭アドレスを入力します。
- 読み出しデータタイプを選択します。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占有）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占有）
  - 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）
- 指定された入力レジスタのアドレスのデータが満足する条件。

## 保持レジスタを待機


**説明:** 指定された保持レジスタのアドレスのデータが条件を満たすと、次のコマンドを実行します。

**パラメータ:**

- 保持レジスタの開始アドレス。
- 読み出しデータタイプを選択します。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占有）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占有）
  - 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）
- 指定された保持レジスタのアドレスのデータが満足する条件。

## 接点レジスタを待機


**説明:** 指定された接点レジスタのアドレスのデータが条件を満たすと、次のコマンドを実行します。

**パラメータ:**

- 接点レジスタの開始アドレス。
- 接点レジスタの指定されたアドレスの値が満たすべき条件。

## コイルレジスタを待機


**説明:** 指定されたコイルレのアドレスのデータが条件を満たすと、次のコマンドを実行します。

**パラメータ:**

- コイルレジスタの開始アドレス。
- 指定されたコイルのアドレスのデータが満足する条件。

## 入力レジスタを読み取る


**説明:** 指定された入力レジスタのアドレスのデータを取得します。

**パラメータ:**

- 入力レジスタの開始アドレス。
- 読み出しデータタイプを選択します。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占有）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占有）
  - 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:** 指定された入力レジスタのアドレスのデータ。

## 保持レジスタを読み取る


**説明:** 指定された保持レジスタのアドレスのデータを取得します。

**パラメータ:**

- 保持レジスタの開始アドレス。
- 読み出しデータタイプを選択します。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占有）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占有）
  - 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:** 指定された保持レジスタのアドレスのデータ。

## 接点レジスタを読み取る


**説明:** 指定された入力ステータスのアドレスのデータを取得します。

**パラメータ:** 接点レジスタの開始アドレス。

**戻り値:** 指定された接点レジスタのアドレスのデータ。

## コイルレジスタを読み取る


**説明:** 指定されたコイルのアドレスのデータを取得します。

**パラメータ:** コイルレジスタの開始アドレス。

**戻り値:** 指定されたコイルレジスタのアドレスのデータ。

## コイルレジスタを連続して読み取る


**説明:** 指定したアドレスから連続してコイルレジスタの値を読み取ります。

**パラメータ:**

- コイルレジスタの開始アドレス。
- 連続して読み取るレジスタのビット数。

**戻り値:** 指定したアドレスからのコイルレジスタの値をテーブル（配列）として返します。テーブル内の最初の値は、開始アドレスのコイルレジスタの値に対応します。

## 保持レジスタを連続して読み取る


**説明:** 指定したアドレスから連続して保持レジスタの値を読み取ります。

**パラメータ:**

- 保持レジスタの開始アドレス。
- 連続して読み取るレジスタの数。
- 読み取るデータの型を選択:
  - U16：16ビット符号なし整数（2バイト、1レジスタを占有）
  - U32：32ビット符号なし整数（4バイト、2レジスタを占有）
  - 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:** 指定したアドレスからの保持レジスタの値をテーブル（配列）として返します。テーブル内の最初の値は、開始アドレスの保持レジスタの値に対応します。

## 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の指定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。
- 書き込む値。0または1のみ指定可能です。

## コイルレジスタに連続して書き込む


**説明:** 指定された値を連続してコイルレジスタの指定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。
- 書き込む値を入力します。複数の値はカンマで区切り、各値は0または1のみ指定可能です。

## 保持ドレジスタに書き込む


**説明:** 指定された値を保持レジスタの指定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。
- 書き込む値を選択します。選択するデータ型と一致する必要があります。
- 書き込むデータ型を選択:
  - **U16**：16ビット符号なし整数（2バイト、1つのレジスタを占有）
  - **U32**：32ビット符号なし整数（4バイト、2つのレジスタを占有）
  - **F32**：32ビット単精度浮動小数点数（4バイト、2つのレジスタを占有）
  - **F64**：64ビット倍精度浮動小数点数（8バイト、4つのレジスタを占有）

## マスターステーションを閉じる

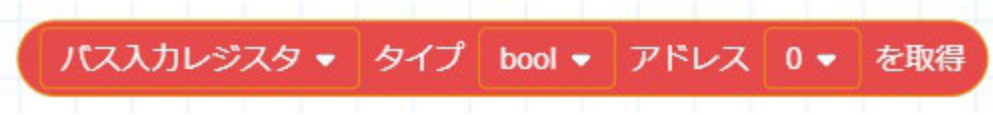

**説明:** 关闭Modbusマスターステーションを閉じ，すべてのスレーブとの接続を切断します。

# 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はこのグループコマンドを対応していません。

## バスレジスタの値を取得する



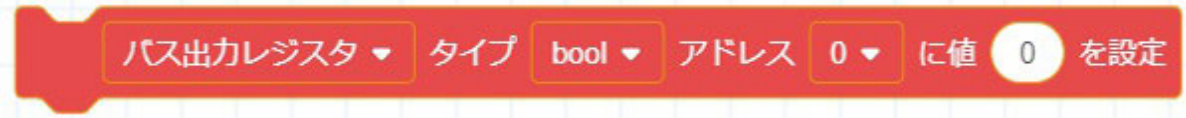

**説明:** 指定されたバスレジスタの値を取得します。

**パラメータ:**

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

**戻り値:** 指定したレジスタの値。 bool型の値は0または1です。

## バスレジスタの値を設定する




**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ます。現在はバス出力レジスタのみを対応します。
- レジスタのデータタイプを選択します。bool、int、floatを対応します。
- 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。bool型の値は0または1です。

# TCPブロックグループ

TCPブロック グループは、TCP通信に関連する操作を実行するために使用されます。対応するDEMOを参照して、関連するコマンドの使用をすぐに体験することができます。

## SOCKETを接続する


**説明:** TCPクライアントを作成します。指定されたTCPサーバーとの接続を確立します。

**パラメータ:**

- SOCKETの番号を選択します。最大4つのTCP通信接続を確立することができます。
- TCPサーバーのIPアドレス。
- TCPサーバーのポート。

## 接続したSOCKETの結果を取得する


**説明:** TCP通信の接続結果を取得します。

**パラメータ:** SOCKETの番号を選択します。

**戻り値:** 接続が成功した場合は0を返し、接続が失敗した場合は1を返します。

## SOCKETを作成する


**説明:** TCPサーバーを作成し、クライアントが接続するのを待ちます。

**パラメータ:**

- SOCKETの番号を選択します。最大4つのTCP通信接続を確立することができます。

- TCPサーバーのIPアドレス。
- TCPサーバーのポート。すでにシステムによって占有されている下記のポートは使用しないでください。サーバーの作成に失敗する可能性があります。

  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

## 作成したSOCKETの結果を取得する


**説明：** TCPサーバーの作成結果を取得します。

**パラメータ：** SOCKETの番号を選択します。

**戻り値：** 作成が成功した場合は0を返し、作成が失敗した場合は1を返します。

## SOCKETを閉じる


**説明：** 指定されたSOCKETを閉じ、通信接続を切断します。

**パラメータ：** SOCKETの番号を選択します。

## 変数を受信する


**説明：** TCP通信で変数を受信して保存します。

**パラメータ：**

- SOCKETの番号を選択します。

- 受信する変数のタイプを選択します。文字列と数値を対応しています。
- 受信したデータを保存するために使用されます。作成した変数ブロックを使用します。
- タイムアウト待つ時間を設定します。0に設定すると、タイムアウトは発生せず、条件が満たされるまで待ちます。

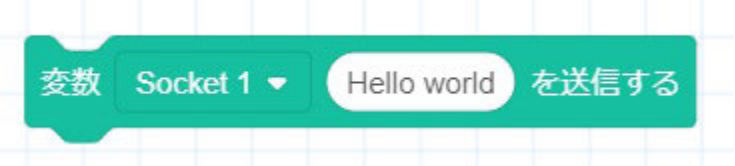
## 受信した変数の結果を取得する


**説明**：受信したTCP変数の結果を取得します。

**パラメータ**：SOCKETの番号を選択します。

**戻り値**：受信が成功した場合は0を返し、受信が失敗した場合は1を返します。

## 変数を送信する




**説明**：TCP通信で変数を送信します。

**パラメータ**：

- SOCKETの番号を選択します。
- 受信するデータ。戻り値が文字列または数値の楕円形ブロックを使用するか直接記入することができます。

## 送信した変数の結果を取得する


**説明**：送信したTCP変数の結果を取得します。

**パラメータ**：SOCKETの番号を選択します。

**戻り値**：送信が成功した場合は0を返し、送信が失敗した場合は1を返します。

# トレイブロックグループ

トレイは、大量な部品を規則的に配置するための積載装置であり、自動積み降ろしプロセスでよく使用されます。通常、トレイには多数の溝が配列状に分布されて、それぞれの溝に部品を配置することができます。

トレイコマンドを使用すると、少数のポイントをティーチングするだけで完全なトレイポイント配列を作成することができます。作成したトレイ内の特定のポイントを取得して、ロボットの自動積み降ろしを迅速に実現することができます。

## トレイを作成する


**説明:** トレイを作成します。1次元、2次元、3次元トレイの作成を対応します。最大20個のトレイを作成することができます。同じ名前のトレイを作成すると、既存のトレイが上書きされ、トレイの数は増えません。

**パラメータ:**

ブロックにトレイ名を入力し、ブロックをクリックすると、ポップアップ設定ウィンドウが表示され、トレイパラメーターを構成します。

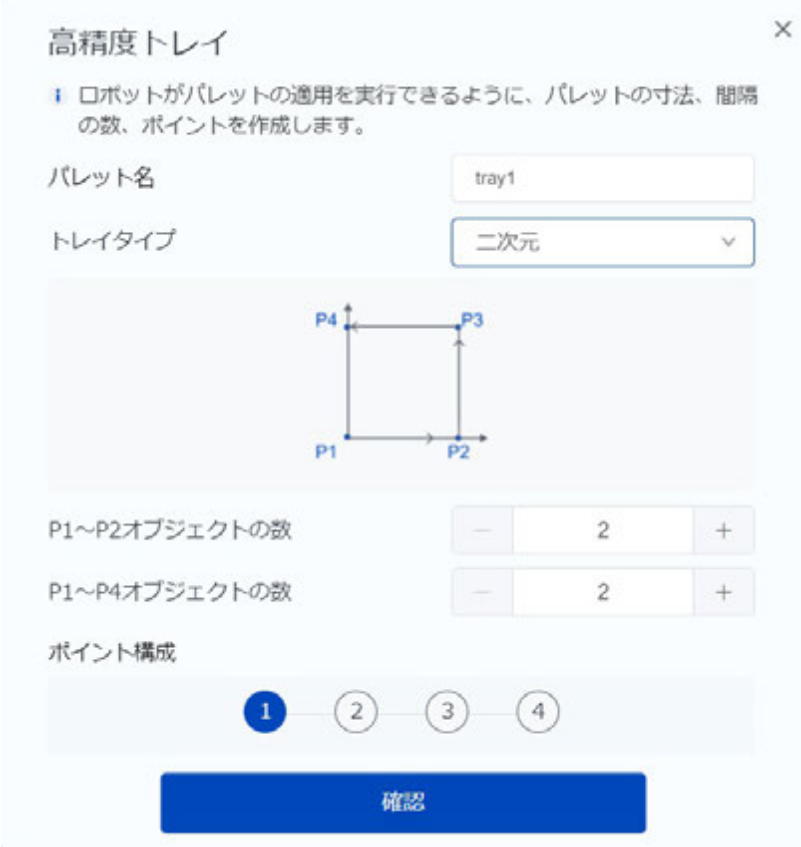
先にトレイの次元を選択します。次元が異なるトレイには、異なるパラメータを設定する必要があります。

- 1次元トレイ


1次元トレイは、直線上に等間隔に配置された一連のポイントの集合です。

- まず、P1～P2のオブジェクトの数を入力します。つまり、1次元トレイ内のポイントの総数になります。
- 次に、P1ポイントとP2ポイントをそれぞれ設定します。保存されたポイントリストからの選択のみが対応可能です。

- 2次元トレイ




2次元トレイは、平面上に配列された一連のポイントの集合です。

- P1～P2（行）とP1～P4（列）のオブジェクトの数をそれぞれ入力します。つまり、この2つの乗積が2次元トレイ内のポイントの総数になります。
- 次に、P1～P4ポイントをそれぞれ設定します。保存されたポイントリストからの選択のみが対応可能です。

- 3次元トレイ


3次元トレイは、空間上に3次元的に分布した一連のポイントの集合です。複数の2次元のトレイを縦に重ね合わせたものとみなすことができます。

- P1～P2（行）、P1～P4（列）、P1～P5（層）のオブジェクトの数をそれぞれ入力します。つまり、この3つの乗積が3次元トレイ内のポイントの総数になります。
- 次に、P1～P8ポイントをそれぞれ設定します。保存されたポイントリストからの選択のみが対応可能です。

⚠**注意:**
末端ツールを使用する場合は、ポイント位置をティーチングする時に必ず末端ツールに対応したツール座標系を選択してください。

## トレイのポイント総数を取得する


**説明:** 作成したトレイのポイント総数を取得します。

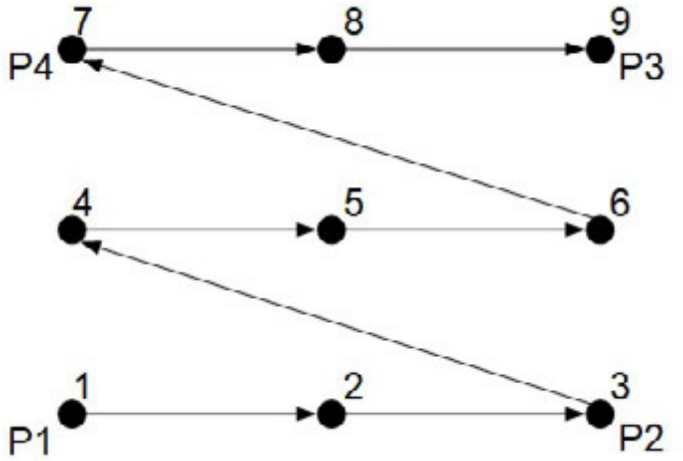
**パラメータ:** 作成したトレイ名を選択します。

**戻り値:** 指定されたトレイのポイント総数。

## トレイのポイントを取得する


**説明:** 指定されたトレイの指定番号のポイントを取得します。ポイントの番号は、トレイを作成した時に設定したポイントの順番に関連します。

- 1次元トレイ：P1ポイントの番号は1、P2ポイントの番号はポイント総数と同じです。以降同様です。
- 2次元トレイ：下図にて3×3のトレイを例として、ティーチングポイントとポイントの番号の関連性を説明します。




- 3次元トレイ：2次元トレイを参考して、2層目の1番目のポイントの番号は1層目の最後のポイントの番号に1を足します。以降同様です。

**パラメータ:**

- 作成したトレイ名を選択します。
- 取得するポイントの番号を入力します。

**戻り値:** 対応するポイントの座標。

# クイックスタート

- Modbusレジスタデータの読み書き
- TCP通信によるデータの転送

# Modbusレジスタデータの読み書き

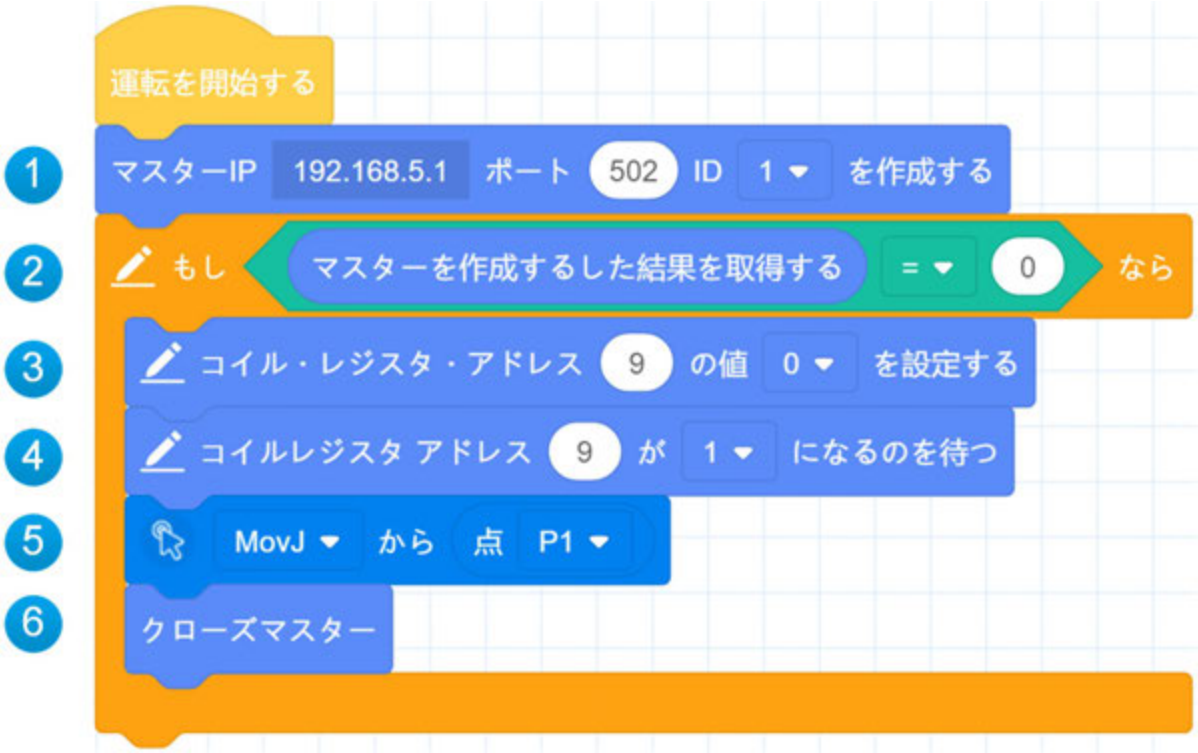
## シーンの説明

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。

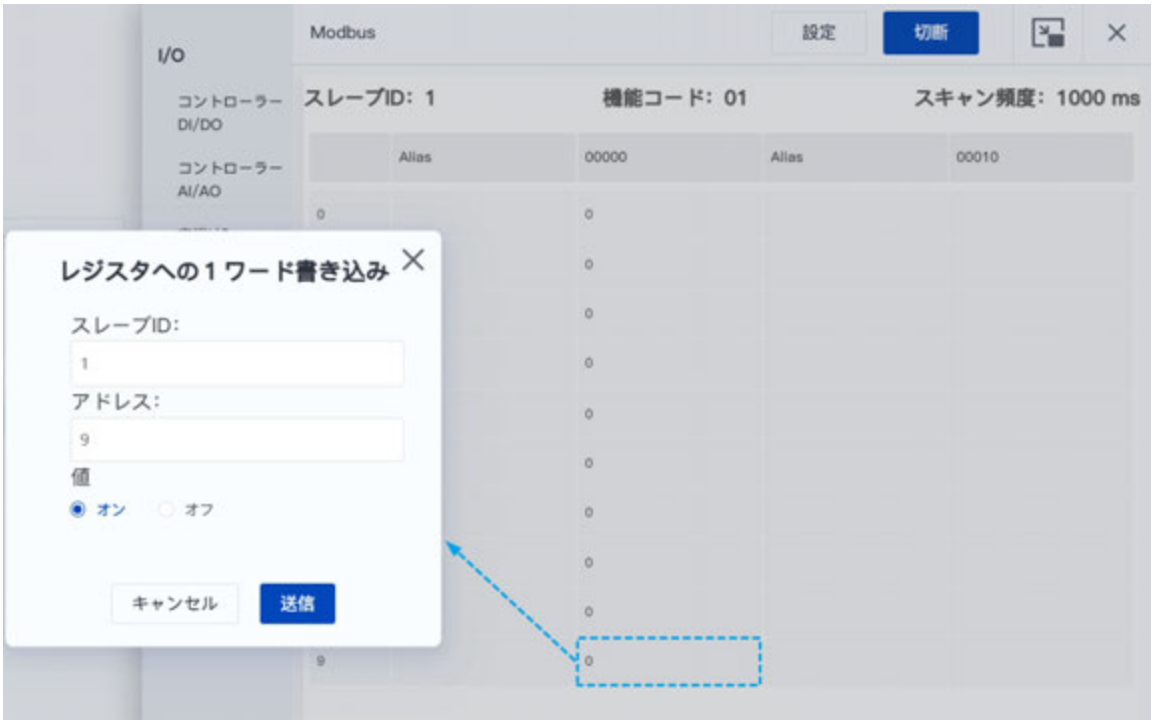
## プログラム実行

迅速にこのプログラムを実行する必要がある場合、Modbusモニタリング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. **モニタリング > Modbus**画面を開き、右上角の**接続**をクリックします。
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**をクリックします。

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**をクリックします。




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。

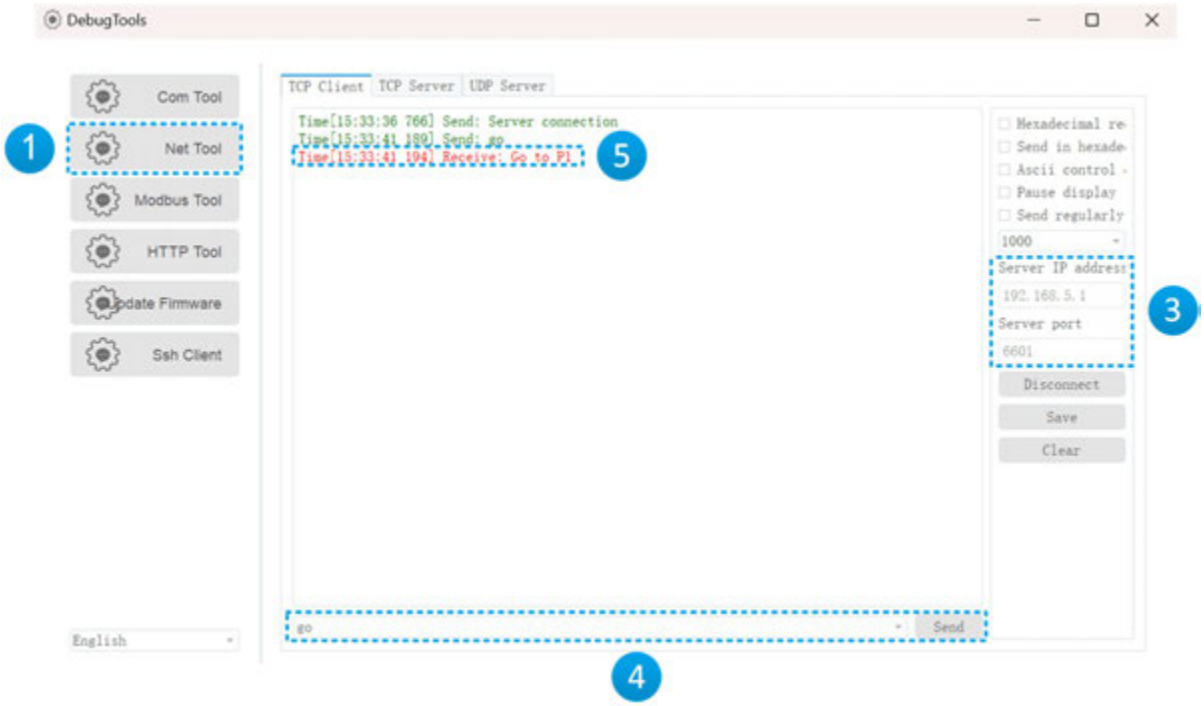

4. 受信した文字列が「go」を含んでいるかを判断します。含んでいる場合、ステップ5と6を実行します。そうでない場合、直接ステップ7を実行します。
5. 文字列「Go to P1」をクライアントに送信します。
6. ロボットを制御してP1点まで移動します。P1はユーザ定義の点です。
7. TPCサーバを終了します。

## プログラム実行

プログラムを迅速に実行する必要がある場合は、DobotStudio Pro にTCPクライアントとして付属しているデバッグ ツールを使用することができます。

1. **設定 > デバッグ ツール** を開き、**Net Tool > TCP Client** 画面に入ります。
2. ロボットを P1 以外のポイントに移動し (ロボットが移動命令を実行したかどうかを後で観察しやすくするため)、プログラムを保存して実行します。
3. 実行ログに TCP サーバーが正常に作成されたことが示されたら、デバッグ ツールでサーバーの IP アドレスとポートを変更し、[接続] をクリックします。
4. 接続が成功したら、デバッグ ツールの下に「go」と入力し、[送信] をクリックします。
5. デバッグ ツールが「Go to P1」メッセージを受信するかどうか、およびロボットが P1 に移動するかどうかを観察します。

下の図は、デバッグ ツールのインターフェイスを示しています。図内のシリアル番号は上記の手順に対応しています。


DebugTools
Com Tool
Net Tool
Modbus Tool
HTTP Tool
Update Firmware
Ssh Client
English
TCP Client
TCP Server
UDP Server
Time[15:33:36 766] Send: Server connection
Time[15:33:41 189] Send: go
Time[15:33:41 194] Receive: Go to P1
Hexadecimal re
Send in hexade
Ascii control
Pause display
Send regularly
1000
Server IP address
192.168.5.1
Server port
6601
Disconnect
Save
Clear
go
Send
1
2
3
4
5

# 付録C スクリプトプログラミング関数の説明


- C.1 基本文法
- C.2 共通説明
- C.3 モーションコマンド
- C.4 相対運動コマンド
- C.5 モーションパラメータ
- C.6 IO
- C.7 末端ツール
- C.8 TCP&UDP
- C.9 Modbus
- C.10 バスレジスタ
- C.11 プログラム制御
- C.12 トレイ
- C.13 セフティスキン

# 基本文法


- 基本概念
- 変数とデータタイプ
- 演算子
- プロセス制御
- 関数
- 数学演算によく使われる関数
- 文字列処理によく使われる関数
- テーブル（配列）操作によく使われる関数

# 基本概念

> **説明:**
> Lua プログラミングに関する知識を体系的に学びたい場合は、インターネットで Lua チュートリアルを検索してください。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 Lua 構文のみがリストされています。

## 識別子

識別子は、変数、関数、またはその他のユーザー定義項目を定義するために使用されます。識別子は、文字 A ~ Z、a ~ z、またはアンダースコア _ で始まり、その後に 0 個以上の文字、アンダースコア、および数字 (0 ~ 9) が続きます。

Lua の予約語にも当てはまるため、アンダースコアと大文字を識別子として使用しないことをお勧めします。

Lua では、@、$、% などの特殊文字を使用して識別子を定義することはできません。

**Lua は大文字と小文字を区別するプログラミング言語であり、プログラム内のすべての識別子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。**

## キーワード

Lua の予約キーワードは以下のとおりです。予約されたキーワードは、定数、変数、またはその他のユーザー定義の識別子として使用できません。

**and, break, do, else, elseif, end, false ,for, function, if , in , local, nil, not, or, repeat, return, then, true, until, while, goto**

一般的な慣例により、アンダースコアで始まり、その後に大文字の文字列が続く名前 (_VERSION など) は、Lua 内部グローバル変数用に予約されています。

## コメント

コメントはプログラムの実行には影響せず、主にコードを読む人がプログラムを理解するのを助けるために使用されます。

### 一行コメント

単一行のコードでは、2 つのマイナス記号以降はコメントとして扱われます。コメントは別の行に置くことも、コードの後に置くこともできます。

```
--一行コメント
print("Hello World! ")  --一行コメント
```

### 複数行のコメント

複数行のコメントは「--[[」で始まり「--]]」で終わり、その間の内容はコメントとして扱われます。

```
--[[
 複数行のコメント
 複数行のコメント
 --]]
```

# 変数とデータタイプ

変数は、パラメータとして渡される値、または結果として返される値を格納するために使用されます。変数には「=」を通して値が代入されます。

Luaでは、localを使用してローカル変数を明示的に宣言します（現在の関数で有効）。ローカル変数のスコープは、宣言位置から関数の最後までです。

localを使用して明示的に宣言されていない変数は、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 ファイルで有効）になり、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です。

さらに、DobotStudio Proの**モニタリング > グローバル変数**画面からコントローラー レベルのグローバル変数を設定することもできます。これは、同じコントローラー内の異なるスクリプトファイルで直接呼び出すことができます。


**例1**：主に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ます。

```
ffunction func()      -- ファンクションを定義し、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(変数スコープ外で呼び出され、nilとして出力されます。以下同様)
print(b)            --> 2
```

**例2**：ローカル変数がスクリプトレベルの変数と同じ名前を持つことができるが、それらは関数内でのみ有効であることを示しています。 do/endなどのプロセス制御用のコードブロック（ループ、条件判定）も関数です。

```
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はローカル変数aを呼び出します。
end

print(a)        --> a，ステートメントブロック外ではスクリプトレベル変数aを呼び出します。
```

**例3**：主にグローバル変数のスコープと他の2種類の変数の違い、およびグローバル変数のグローバル保持関数を説明します。

```
-- 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がグローバル変数画面に追加されたと仮定します。
-- g1はグローバル保存しない、値は10。
-- g2はグローバル保存する、値は20

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

print(g1)           --> 10
print(g2)           --> 20
SetGlobalVariable("g1",11)  -- グローバル保存しないグローバル変数に割り当て
SetGlobalVariable("g2",22)  -- グローバル保存するグローバル変数に割り当て
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の後に実行されます

print(a)        --> nil
print(b)        --> nil
print(g1)       --> 10(グローバル保持しない変数は、プロジェクト1での変更前の値に復元されます。)
print(g2)       --> 22(グローバル保持する変数は、プロジェクト1での変更後の値になります。)
```

変数名には、Luaによって予約されているキーワードを除き、数字で始まらない文字、アンダースコア、および数字で構成される任意の文字列を使用できます。

Lua変数には型の定義は必要ありません。変数に値を割り当てるだけでよく、Luaは値に基づいて変数の型を自動的に決定します。スクリプトで割り当てられた変数にさまざまなタイプの値を割り当てると、変数のタイプが変更されます。ただし、**モニタリング > グローバル変数** ページ

で設定された変数のタイプは変更できず、割り当てのタイプを変更する必要があります。変数の作成時に選択されたタイプと一致しない場合、コントローラーはスクリプトの実行時にエラーを報告します。

Luaはさまざまなデータ型をサポートしており、より一般的なものには数値、ブール値、文字列、テーブルなどがあります。 Lua の配列はテーブルの一種です。

Luaには nil と呼ばれる特別なデータ型もあります。nil は空 (有効な値がない) を意味します。たとえば、値が割り当てられていない変数を出力すると、nil 値が出力されます。

## 数字

LuaのNumberは倍精度型の実浮動小数点数であり、以下のような記述方法が数値として扱われます。

- 2
- 2.2
- 0.2
- 2e+1
- 0.2e-1
- 7.8263692594256e-06

## ブール値

ブール型にはtrue（true）とfalse（false）の2つのオプション値しかありません。Luaはfalseとnilをfalseとして扱い、それ以外はすべてtrue、数値0も true として扱います。

## 文字列

stringは、数字、文字、アンダースコアで構成される文字列です。stringは下記の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。

数値文字列に対して算術演算を実行する場合、Lua は数値文字列を数値に変換しようとします。

```
-- シングルクォートで文字列を定義
local str1 = 'Hello, Lua!'
print(str1)  -- 出力: Hello, Lua!

-- ダブルクォートで文字列を定義
local str2 = "Hello, Lua!"
print(str2)  -- 出力: Hello, Lua!

-- [[ と ]] を使用して複数行の文字列を定義
```

```lua
local str3 = [[
This is a multi-line
string in Lua.
]]
print(str3)
-- 出力:
-- This is a multi-line
-- string in Lua.

-- Luaは数字の文字列を自動的に数値に変換して算術演算を実行
local numStr = "10"
local result = numStr + 20  -- "10" を数値 10 に自動変換
print(result)  -- 出力: 30
```

## 表

テーブルはインデックス付きデータのセットです。

- 最も簡単な作成方法は {} で、これにより空のテーブルを作成できます。また、直接初期化してテーブルを作成することもできます。
- テーブルは実際には関連配列であり、任意の型の値をキーとして使用できますが、その値は nil であってはなりません。
- テーブルはサイズが固定されておらず、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。

```lua
-- 連続したインデックスを持つテーブルを初期化
local tbl = {[1] = 2, [2] = 6, [3] = 34, [4] = 5}
print("tbl の長さ ", #tbl)  -- 出力: 4
```

この例では、`tbl` のインデックス `[1]` から `[4]` は連続しているため、`#tbl` は正しく長さ `4` を返します。

## 配列

配列とは、同じデータ型の要素を一定の順序で並べた集合で、一次元配列や多次元配列があります。

Luaでは、配列は `table` 型に属し、インデックスキーは整数で表すことができ、配列のサイズは固定されていません。

- 一次元配列：最も単純な配列であり、その論理構造は線形リストです。
- 多次元配列：配列の中に配列が含まれている、または一次元配列のインデックスキーが別の配列を指しているものです。

Luaでは、配列のインデックス値はデフォルトで1から始まりますが、0や負の値から始めることも指定できます。存在しないインデックスで配列要素にアクセスすると、`nil` が返されます。

**例1**：一次元配列は、forループを使って配列内の要素を取得できます。

```
-- 一次元配列を作成
local array = {"Lua", "Tutorial"}

-- forループを使用して配列を走査、インデックス1から開始
for i = 1, #array do
    print(array[i])    -- 出力: Lua Tutorial
end
```

**例2**：数値要素を含む一次元配列

```
-- 数値要素を持つ一次元配列を作成
local numbers = {10, 20, 30, 40, 50}

-- forループを使用して配列を走査、インデックス1から開始
for i = 1, #numbers do
    print(numbers[i])    -- 出力: 10 20 30 40 50
end
```

**例3**：カスタムインデックスの配列

```
-- 配列を作成し、負の値と正の値をインデックスとして使用
local array = {}
for i = -2, 2 do
    array[i] = i * 2 + 1  -- 配列に値を代入
end

-- forループを使用して配列を走査、インデックス -2 から 2 まで
for i = -2, 2 do
    print(array[i])  -- 出力: -3 -1 1 3 5
end
```

**例4**：3行3列の配列を表す多次元配列

```
--- 配列を初期化
array = {}
for i = 1, 3 do
    array[i] = {}
    for j = 1, 3 do
        array[i][j] = i * j
    end
end

-- 配列にアクセス
for i = 1, 3 do
    for j = 1, 3 do
        print(array[i][j])     -- 出力結果は順に：1 2 3 2 4 6 3 6 9
    end
end
```

# 演算子

## 算術演算子

| コマンド記号 | 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 |

例:

```
a = 20
b = 5
print(a + b)        -- aとbの和を出力：25
print(a - b)        -- aからbを引いた結果を出力：15
print(a * b)        -- aとbの積を出力：100
print(a / b)        -- aをbで割った結果を出力：4
print(a // b)       -- aをbで割った商（整数部分）を出力：4
print(a % b)        -- aをbで割った余りを出力：0
print(a ^ b)        -- aのb乗の結果を出力：3200000
```

## ビット演算子

| コマンド記号 | 説明 |
| --- | --- |
| & | 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:

```
print(a & b)        -- aとbのビットごとのANDの結果を出力：4
```

```
print(a | b)        -- aとbのビットごとのORの結果を出力：21
print(a ~ b)        -- aとbのビットごとのXORの結果を出力：17
print(a << b)       -- aをbビット左シフトした結果を出力：640
print(a >> b)       -- aをbビット右シフトした結果を出力：0
```

## 関係演算子

| コマンド記号 | 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
b = 5               -- 変数bを作成
print(a == b)       -- aがbと等しいかを比較して出力：false
print(a ~= b)       -- aがbと等しくないかを比較して出力：true
print(a <= b)       -- aがb以下かを比較して出力：false
print(a >= b)       -- aがb以上かを比較して出力：true
print(a < b)        -- aがbより小さいかを比較して出力：false
print(a > b)        -- aがbより大きいかを比較して出力：true
```

## ロジック演算子

| コマンド記号 | 説明 |
| --- | --- |
| and | 論理積（AND）演算子。両側がtrueの場合のみ結果がtrueになります。一方でもfalseであれば結果はfalseです。 |
| or | 論理和（OR）演算子。一方でも結果がtrueであれば結果はtrueになります。両側がfalseの場合のみ結果はfalseです。 |
| not | 論理否定（NOT）演算子。判定結果を反転させます。 |

```
a = true
b = false
print(a and b)        -- 出力：false。aとbの両方がtrueの場合のみtrueを返す
print(a or b)         -- 出力：true。aまたはbのいずれかがtrueであればtrueを返す
print(not a)          -- 出力：false。aがtrueのため、反転してfalseになる
```

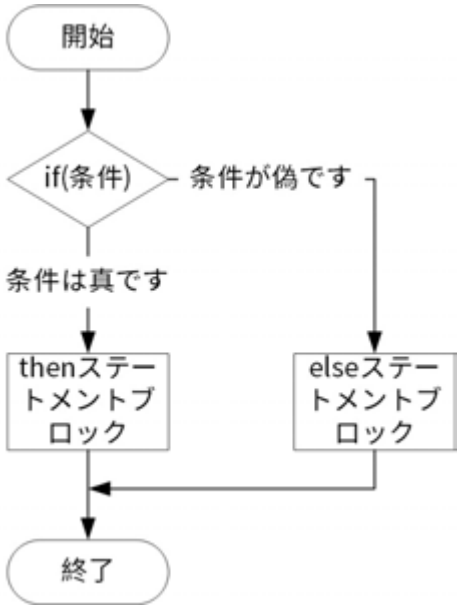
```
print(not b)          -- 出力: true。bがfalseのため、反転してtrueになる
print(not (20 > 5))   -- 出力: false。20 > 5はtrueのため、反転してfalseになる
```

# プロセス制御

| 命令記号 | 説明 |
| --- | --- |
| if...<br>then...<br>else...<br>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...<br>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...<br>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...<br>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。 |

## if条件判断指令

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。



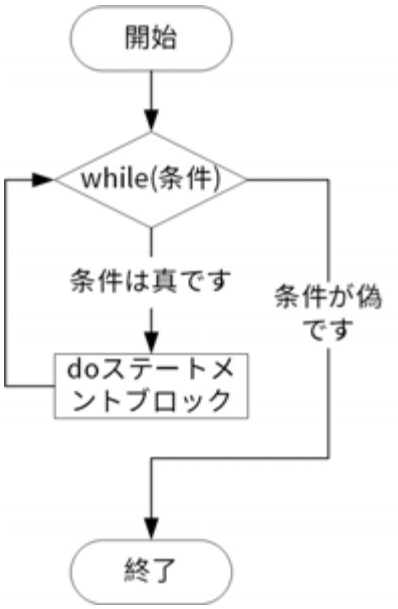

ifステートメントは入れ子にすることができます。典型的な例を次に示します。

```
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
```

## 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:

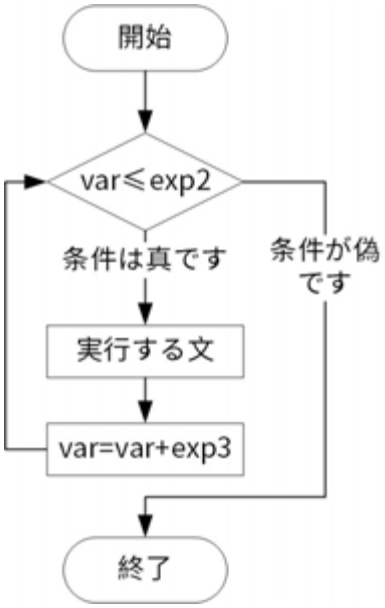
```
a=10
while( a < 20 )
do
    print("aの値は次のとおりです: ", a)  -- 10回実行すると出力値は10～19になります
    a = a+1
end
```

## 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。

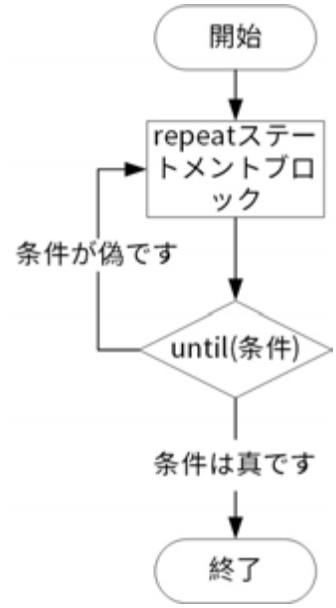



例:

```
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
```

## 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)
```

# 関数

関数は、文や式を抽象化するための主要な方法です。その定義形式は次のとおりです:

```
function function_name(argument1, argument2, argument3..., argumentn)
    function_body
    return result_params_comma_separated
end
```

- **function_name**: 関数名。関数は、事前に定義してその名前を使って呼び出すことも、呼び出し時に直接定義することもできます。後者の場合、関数名を省略することが可能です。
- **argument1, argument2, argument3..., argumentn**: 関数のパラメータ。複数のパラメータはカンマで区切ります。関数はパラメータを持たないこともできます。
- **function_body**: 関数本体。関数内で実行されるコードブロックです。
- **result_params_comma_separated**: 関数の戻り値。Lua言語の関数は複数の値を返すことができ、それぞれをカンマで区切ります。関数が戻り値を持たない場合もあります。

**例 1**: パラメータや戻り値を持たない関数

```
function greet()
    print("Hello, Lua!")  -- 関数本体
end

greet()  -- 関数を呼び出し、出力: Hello, Lua!
```

**例 2**: パラメータを持つが、戻り値を持たない関数

```
function printSquare(number)
    print("Square of " .. number .. " is " .. (number * number))  -- 関数本体
end

printSquare(5)  -- 関数を呼び出し、出力: Square of 5 is 25
```

**例 3**: パラメータと戻り値を持つ関数

```
function maximum(a)
    local mi = 1  -- 最大値のインデックス
    local m = a[mi]  -- 最大値
    for i, val in ipairs(a) do
        if val > m then
            mi = i
            m = val
        end
    end
    return m, mi  -- 最大値とそのインデックスを返す
```

```
end

local maxVal, maxIndex = maximum({8, 10, 23, 12, 5})
print("Max value:", maxVal)  -- 出力: Max value: 23
print("Index of max value:", maxIndex)  -- 出力: Index of max value: 3
```

**説明:**
本マニュアルでは、DOBOTで事前定義された関数の使用方法について説明します。ユーザーは独自に関数を定義することも可能です。独自に関数を定義する場合、関数の定義を `global.lua` ファイルまたはその関数を呼び出す `src*.lua` ファイルの先頭に記述してください。そうしないと、実行時にエラーが発生します。

# 数学演算によく使われる関数

Luaは基本的な数学関数を提供しており、算術演算子を補完します。

## math.abs(X)

Xの絶対値を返します。例:

```
print("math.abs(-10):", math.abs(-10))  -- 出力: 10
```

## math.floor(X)

例：X以下の最大の整数値を返します（切り捨て）。例:

```
print("math.floor(3.7):", math.floor(3.7))  -- 出力: 3
```

## math.ceil(X)

X以上の最小の整数値（切り上げ）を返します。例:

```
print("math.ceil(3.2):", math.ceil(3.2))  -- 出力: 4
```

## math.sqrt(X)

Xのルート（平方根）を返します。例:

```
print("math.sqrt(16):", math.sqrt(16))  -- 出力: 4
```

## math.rad(X)

角度Xのラジアン値を返します（角度をラジアンに変換）。例:

```
print("math.rad(180):", math.rad(180))  -- 出力: 3.1415926535898
```

## math.deg(X)

ラジアンXの角度値を返します（ラジアンを角度に変換）。例:

```
print("math.deg(math.pi):", math.deg(math.pi))  -- 出力: 180
```

## math.sin(X)

ラジアンXの正弦値を返します。

角度をパラメータとして使用したい場合は、math.radを使用して変換できます。例:

```
print("math.sin(math.rad(30)):", math.sin(math.rad(30)))  -- 出力: 0.5
```

## math.cos(X)

ラジアンXの余弦値を返します。

角度をパラメータとして使用したい場合は、math.radを使用して変換できます。例:

```
print("math.cos(math.rad(60)):", math.cos(math.rad(60)))  -- 出力: 0.5
```

## math.tan(X)

ラジアンXの正接値を返します。

角度をパラメータとして使用したい場合は、math.radを使用して変換できます。例:

```
print("math.tan(math.rad(45)):", math.tan(math.rad(45)))  -- 出力: 1
```

## math.asin(X)

Xのアークサイン値を返します。戻り値はラジアンであり、math.degを使用して角度に変換できます。例:

```
print("math.deg(math.asin(0.5)):", math.deg(math.asin(0.5)))  -- 出力: 30，戻り値は角度
```

## math.acos(X)

Xのアークコサイン値を返します。戻り値はラジアンであり、math.degを使用して角度に変換できます。例:

```
print("math.deg(math.acos(0.5)):", math.deg(math.acos(0.5)))  -- 出力: 60，戻り値は角度
```

## math.atan(X)

Xのアークタンジェント値を返します。戻り値はラジアンであり、math.degを使用して角度に変換できます。例:

```
print("math.deg(math.atan(1)):", math.deg(math.atan(1)))  -- 出力: 45，戻り値は角度
```

## math.log(X)

Xの自然対数を返します。例:

```
print("math.log(10):", math.log(10))  -- 出力: 2.302585092994
```

## math.exp(X)

自然定数eのX乗を返します。例:

```
print("math.exp(1):", math.exp(1))  -- 出力: 2.718281828459
```

# 文字列処理によく使われる関数

Luaは、文字列の検索や置換などの操作を実行できる文字列処理用の一般的な関数を提供します。

## string.sub(s, i, j)

文字列をインターセプトするために使用されます。

**必須パラメータ**

- s：インターセプトする文字列。
- i：インターセプトの開始位置、1から始まります。

**オプションパラメータ**

- j：インターセプトの終了位置。デフォルトは -1 で、最後の文字を示します。

**戻り値**

取得した文字列をインターセプトします。

**例**

```
sub1 = string.sub("abcde", 3)  --3位からインターセプトする：cde
sub2 = string.sub("abcde", 1, 3) --1桁目から3桁目までインターセプトする：abc
sub3 = string.sub("abcde", 3, 3) --3桁目をインターセプトする：c
```

## string.find (s, sub, i, plain)

指定された文字列内の部分文字列を検索し、見つかったインデックスを返します。

**必須パラメータ**

- s：検索されている文字列。
- sub：検索する部分文字列。

**選択可能なパラメータ**

- i：検索を開始する位置、デフォルトは 1。
- plain：プレーンテキストモードを使用するかどうか。このパラメータを指定する場合は、パラメータ i も指定する必要があります。
  - true：プレーン テキストを使用します。この場合、subはプレーン テキスト文字列として扱われます。。
  - false：デフォルト値、 パターンマッチングメカニズムを対応します。

**戻り値**

- 元の文字列内で最初に一致した部分文字列の開始インデックスと終了インデックス。

- スキーマでキャプチャが定義されている場合、キャプチャされた値は 2 つのインデックス後に返されます。
- 部分文字列が見つからない場合はnilを返します。

**例**

```
i,j = string.find("abcde", "cd")  --abcdeでcdを検索し、インデックスを返します：i为3，j为4

if string.find(str1, str2) ~= nil --str1にstr2が含まれる場合、TODOの内容を実行します。
  then
  --TODO
  end

--高度な使用方法
i,j = string.find("abcabc", "ab", 3)  --abcdeで位置3から始まるcdを検索し、インデックスを返します：i为4，j为5

i,j = string.find("123%abc", "%a")  --パターン マッチング メカニズムはデフォルトでサポートされており、パラメーター%aはワイルドカード（任意の文字を表す）として解釈されるため、元の文字列の最初の文字a、i 、 jは5です。
i,j = string.find("123%abc", "%a", 1, true)  --プレーン テキスト モードを使用すると、パラメーター%aはプレーン テキストとみなされ、元の文字列の%aと一致、iは4、jは5。

i,j,sub = string.find("abc 10 edf 100", "(%d+)")  --最初の数値文字列、Returnvalueで見つかったインデックス、およびキャプチャされた結果を見つけてキャプチャします：iは5、jは6、subは10。
```

## string.match(s, sub, i)

指定された文字列内の部分文字列を検索し、キャプチャ結果またはパターン一致の部分文字列を返します。

**必須パラメータ**

- s：検索されている文字列。
- sub：検索する部分文字列。 パターンマッチングメカニズムを対応します。

**選択可能なパラメータ**

- i：検索を開始する位置。デフォルトは 1 です。

**戻り値**

- 一致した部分文字列を返します。
- スキーマでキャプチャが定義されている場合、すべてのキャプチャ結果が返されます。
- 部分文字列が見つからない場合はnilを返します。

**例**

```
sub1 = string.match("abcde", "cd")  --abcdeでcdを検索し、見つかった部分文字列を返します：cd

sub2 = string.match("abc 10 edf 100", "%a+", 4)  --%a+は連続する文字を表し、4番目の位置から検索が開始され、返される一致結果は次のとおりです：edf
```

```
sub3 = string.match("abc 10 edf 100", "%a+ (%d+) %a+")  --2つの文字列の間の数値をキャプチャし、キャプチャした結果を返します: 10
```

## string.gmatch(s, sub)

指定された文字列をループして部分文字列を検索し、キャプチャ結果または部分文字列に一致するパターンを返します。

**必須パラメータ**

- s：検索されている文字列。
- sub：検索する部分文字列。 パターンマッチングメカニズムを対応します。

**戻り値**

- イテレータ関数を返します。この関数が呼び出されるたびに、一致する結果が返されます。
- 一致する値が見つからない場合、反復関数はnilを返します。

**例**

```
for word in string.gmatch("Hello world from dobot", "%a+")
do
    print(word)
end
--[[
 実行中の効果は、各単語を1行ずつ出力します:
 Hello
 world
 from
 dobot
 --]]
```

## string.gsub(s, find, repl, n)

文字列の指定された部分を置換するために使用されます。

**必須パラメータ**

- s：検索されている文字列。
- find：置き換えられる文字。 パターンマッチングメカニズムを対応します。
- repl：置き換える文字。

**選択可能なパラメータ**

- n：置換の最大数。ご指定のない場合は全て交換させていただきます。

**戻り値**

置換された文字列と置換の数。

**例**

```
str, n = string.gsub("aaaa","a","z"); --すべてのaをzに置き換え。戻り値：strはzzzz，nは4

str, n = string.gsub("aaaa","a","z",3); --最初の3つのaをzに置き換え、戻り値：strはzzza，nは3

str, n = string.gsub("a1b2c3","%d","z"); --すべての数字をzに置き換え、戻り値：strはazbzcz，nは3
```

## パターンマッチングメカニズム

Lua のパターンマッチングメカニズムは正規表現に似ており、**string.find、string.gmatch、string.gsub、string.match** によるあいまいマッチングを実現するために使用できます。

### 文字クラス

文字クラスはパターン マッチングの基本単位であり、特定の文字セットを表すために使用されます。主要な文字のクラスは次のとおりです:

- 単一の文字( `^$()%.[]\*+-?` 以外): 文字そのものと一致します。
- `.`: 単一のドットは、任意の文字と一致することを意味します。
- `%a`: 任意の文字と一致します。
- `%c`: 任意の制御文字と一致します (例: `\n` )。
- `%d`: 任意の数値と一致します。
- `%l`: 任意の小文字と一致します。
- `%p`: 任意の句読点と一致します。
- `%s`: 空白文字 (スペース) と一致します。
- `%u`: 任意の大文字と一致します。
- `%w`: 任意の文字/数字と一致します。
- `%x`: 任意の 16 進数と一致します。
- `%x` (x は英数字以外の文字): 文字 x と一致します。主に特殊文字 ( `^$()%.[]*+-?` ) の一致を処理するために使用されます。 、「% %」などは「%」と一致します。
- `[複数の文字クラス]`: `[]` に含まれる任意の文字クラスと一致します。たとえば、 `[%w_]` は任意の文字/数字 ( `%w` ) またはアンダースコア ( `_` ) に一致します。
- `[^複数の文字クラス]`: `[]` に含まれない任意の文字クラスと一致します。 たとえば、 `[^%s%p]` は空白文字や句読点以外の文字に一致します。

単一の文字で表されるすべてのカテゴリ ( `%a` 、 `%c` など) は、文字が大文字に変更されると、対応する補数を表します。 たとえば、「%S」はスペース以外のすべての文字を表します。

### モードエントリー

パターン エントリは、文字クラスと、その文字クラスが一致するパターンを指定する特殊記号の組み合わせです。一般的に使用されるモード エントリは次のとおりです。

- 単一の文字クラスは、そのクラス内の任意の単一文字と一致します。
- `*` が後に続く単一の文字クラスは、そのクラスの 0 個以上の文字と一致します。 このエントリは常に可能な限り長い文字列と一致します。
- 単一の文字クラスの後に「+」を付けると、そのクラスの 1 つ以上の文字と一致します。

このエントリは常に可能な限り長い文字列と一致します。
- `-` が後に続く 1 つの文字クラスは、そのクラスの 0 個以上の文字と一致します。 `*` とは異なり、このエントリは常に可能な限り短い文字列と一致します。
- `?` が後に続く単一の文字クラスは、そのクラスの 0 文字または 1 文字と一致します。
- `%n` 、n は 1 ~ 9 の整数を表し、このエントリはキャプチャ番号 n に等しい部分文字列と一致します (詳細については、以下の **キャプチャ** を参照)。

**モデル**

パターンは、パターン エントリのシーケンスです。例えば:

- dd/mm/yyyy 形式で日付を照合するには、パターン `%d%d/%d%d/%d%d%d%d` を使用します。
- 最初の文字が大文字である単語に一致するには、パターン `%u%l+` を使用します。

**キャプチャ**

パターンには括弧で囲まれたサブパターンを含めることができます。これらのサブパターンはキャッチと呼ばれます。

一致が成功すると、キャプチャされたオブジェクトと一致した部分文字列が保存され、後続の操作に使用されます。 キャプチャには、左括弧の順序で番号が付けられます。 たとえば、パターン `(a*(.)%w(%s*))` の場合:

- 文字列内の `a*(.)%w(%s*)` に一致する文字列はキャプチャ番号 1 です。
- 「 `.` 」に一致する文字はキャプチャ番号 2 です。
- 「 `%s*` 」に一致する文字列はキャプチャ番号 3 です。
- Catch No. 2 と Catch No. 3 は、Catch No. 1 の部分文字列です。

特殊なケースとして、空の `()` は文字列内の対応する文字の位置を取得します。 たとえば、パターン `()aa()` が文字列 `flaaap` に適用される場合、 3 と 5 の 2 つのキャプチャが生成されます。

## その他のよく使われる方法

| 方法 | 説明 |
|---|---|
| string.upper (argument) | すべての文字列を大文字に変換します。 |
| string.lower (argument) | すべての文字列を小文字に変換します。 |
| string.reverse(arg) | 文字列の反転 |
| string.format(...) | printf のような形式の文字列を返します。 |
| string.len(arg) | 文字列の長さを計算する |
| string.rep(string, n) | 文字列 string のコピーを n 個返します |

**例:**

```
str = "Lua"
```

```
print(string.upper(str))         --すべての文字列を大文字に変換し、結果を出力します。: LUA
print(string.lower(str))         --すべての文字列を小文字に変換し、結果を出力します。: lua
print(string.reverse(str))     --文字列を反転して結果を出力する: auL
print(string.len("abc"))         --文字列 abc の長さを計算し、結果を出力します: 3
print(string.format("the value is: %d",4))      --結果を出力します: the value is:4
print(string.rep(str,2))         --文字列を 2 回コピーし、結果を出力します。: LuaLua
```

## その他の演算記号

| コマンド記号 | 説明 |
| --- | --- |
| .. | 2つの文字列を結合します |
| # | 文字列またはテーブルの長さを返します |

```
a = "Hello "
b = "World"
c = {1,2,3}

print(a..b ) --aとbを結合した文字列を出力: Hello World

print(#b) --文字列bの長さを出力: 5

print(#c) --テーブルcの長さを出力:3
```

# テーブル（配列）操作によく使われる関数

Luaは、テーブルの挿入、並べ替え、その他の操作を実行できるテーブル（配列）処理のための基本的な関数を提供します。以下のコマンドで操作するテーブルは連続した1要素の配列である必要があり、インデックスはLuaのデフォルトの1～n（nは配列長）です。

## table.concat (table, sep, start, end)

テーブル内の要素をインデックス順に文字列に連結し、区切り文字と開始/終了位置の指定をサポートします。

**必須パラメータ**

- table：操作対象のテーブル。

**選択可能なパラメータ**

- sep：セパレータ。デフォルト値はセパレータなしです。
- start：開始位置。デフォルト値は 1 です。
- end：終了位置。デフォルト値は配列の長さです。

**戻り値**

連結して得られる文字列。開始位置が終了位置より大きい場合は、空の文字列が返されます。

**例**

```
fruits = {"banana","orange","apple"}

print(table.concat(fruits)) --デフォルトの方法を使用して接続し、結果を出力します：bananaorangeapple

print(table.concat(fruits,",")) --セパレータ接続を指定して結果を出力する：banana,orange,apple

print(table.concat(fruits,",", 2,3)) --セパレータとインデックス接続を指定し、結果を出力する：orange
,apple
```

## table.insert (table, pos, value)

指定された要素をテーブル内の指定された位置に挿入します。

**必須パラメータ**

- table：操作対象のテーブル。
- value：挿入する要素。

**選択可能なパラメータ**

- pos：挿入位置。デフォルト値は配列の長さに1を加えたもので、テーブルの最後に挿入さ

れます。

**例**

```
fruits = {"banana","orange","apple"}

table.insert(fruits,"mango") --最後に挿入する
print(fruits[4])  --結果を出力します：mango

table.insert(fruits,2,"grapes") --2番目のインデックスに挿入
print(fruits[2],",",fruits[3]) --結果を出力します：grapes,orange
```

## table.remove(table, pos)

テーブル内の指定された位置にある要素を削除し、削除された値を返します。

**必須パラメータ**

- table：操作対象のテーブル。

**選択可能なパラメータ**

- pos：挿入位置。デフォルト値は配列の長さです。つまり、テーブルの最後に挿入されます。

**戻り値**

削除される値。

**例**

```
fruits = {"banana","orange","apple"}

print(table.remove(fruits)) --最後の要素を削除し、削除された要素を出力します：apple

print(table.remove(fruits),2) --2 番目の要素を削除し、削除された要素を出力します：orange
```

## table.sort(table, comp)

テーブル内の要素を並べ替えるには、並べ替え方法を指定できます。

**必須パラメータ**

- table：操作対象のテーブル。

**選択可能なパラメータ**

- comp：並べ替え方法。パラメーターは、下記の要件を満たす関数である必要があります：
  - 2つのテーブル要素をパラメータとして受け取ることができます。
  - 2番目のパラメーターの前に最初のパラメーターが必要な場合はtrueを返し、それ以外の場合はfalseを返します。

デフォルトでは、Lua標準演算子「<」が並べ替え方法として使用されます。

**例**

```
fruits = {"banana","orange","apple","grapes"}

print("排序前")
for k,v in ipairs(fruits) do
    print(v)         --結果を出力します: banana orange apple grapes
end

--昇順に並べ替えます
table.sort(fruits)
print("排序后")
for k,v in ipairs(fruits) do
    print(v)         --結果を出力します: apple banana grapes orange
end

--降順に並べ替えます
table.sort(fruits,function(a,b)
    return a > b
end)
print("排序后")
for k,v in ipairs(fruits) do
    print(v)         --結果を出力します: orange grapes banana apple
end
```

# 共通説明

## 運動方式

ロボットの運動方法は以下に分類されます。

**関節運動**

ロボットは現在点の関節角度と目標点の関節角度の差に基づいて、各関節が同時に動作を完了するように計画します。ジョイントモーションはTCP（Tool Center Point）の動作軌道を制約しません。一般的には動作軌道は非直線となります。


関節運転は特異点によって制限されません （特異点の位置の詳細については、ロボットの対応するハードウェア マニュアルをご参照ください）。そのため、動作軌道の条件がない場合、または目標点が特異点に近い場合は、ジョイントモーションを使用することを推奨します。

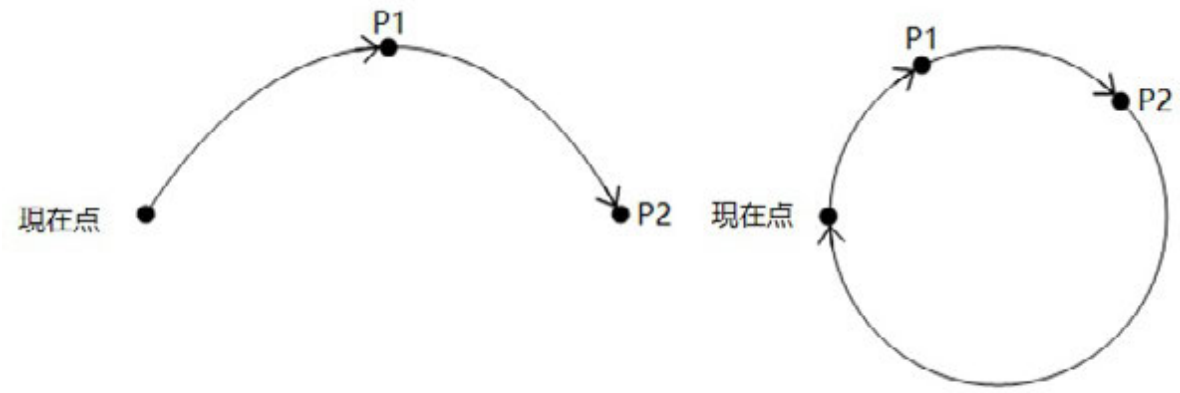
**直線運動**

ロボットは現在点の姿勢と目標点の姿勢に基づいて、TCPの動作軌跡が運動中に等速で変化するように計画します。動作軌道は直線となります。


動作軌跡が特異点を通過する場合、ロボットに直線運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

**円弧運動**

ロボットは、現在点、P1、P2、3つの一直線ではない点によって、円弧または全円を確定します。運動中、ロボットの先端姿勢は、現在点とP2点の姿勢を補間して計算されます。P1点の姿勢は計算に含まれません（つまり、運動中にP1点に到達したときのロボットの姿勢はティーチング時の姿勢と異なる可能性があります）。




動作軌跡が特異点を通過する場合、ロボットに円弧運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

## 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種類の表現方法を対応します。

- 関節変数：各ロボットの各関節の角度(j1-j6)を使用して目標位置を表示します。

  ジョイント変数が直線または円弧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。

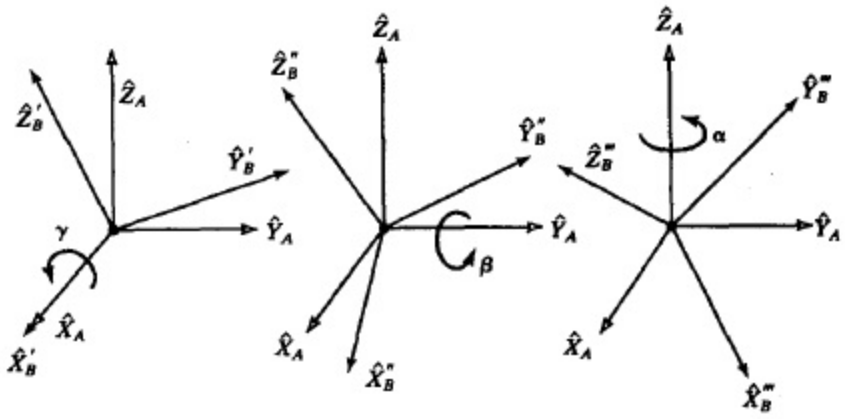
```
{joint = {j1, j2, j3, j4, j5, j6} }
```

- 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ツール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。

  ポーズ変数が関節運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（ロボットの現在の関節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（ユーザー座標系）を中心に旋回します（rx=γ，ry=β，rz=α）。



順序が確定した後、回転行列（ここで、cαはcosα、sαはsinαの省略形、それ以外も同様）を方程式として導出することができます。

$$
\begin{aligned}
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
\end{aligned}
$$

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

この方程式により、ロボット末端の姿勢を計算します。

- ティーチングポイント：制御ソフトウェアを使用して、ティーチング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：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

モーションコマンドのオプションパラメータである `user` と `tool` は、目標点のユーザー座標系およびツール座標系を指定するために使用されます。現在のところ、インデックス番号でのみ指定が可能であり、事前に制御ソフトウェアで対応する座標系を追加する必要があります。

システムがポイントの座標系を選択する優先順位は以下の通りです:

1. モーションコマンドのオプションパラメータで座標系が指定されている場合、指定された座標系を使用します。ポイントパラメータがティーチポイントの場合、ティーチポイントの姿勢座標を指定された座標系の値に換算して使用します。
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:
    - ポイントがティーチポイントの場合、そのティーチポイントに付随する座標系インデックスを使用します。
    - ポイントがジョイント変数または姿勢変数の場合、モーションパラメータで設定されたグローバル座標系を使用します（詳細はUserおよびToolコマンドを参照してください。コマンドで設定されていない場合、デフォルト座標系は0です）。

> **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。
> - 関節運動コマンド（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。

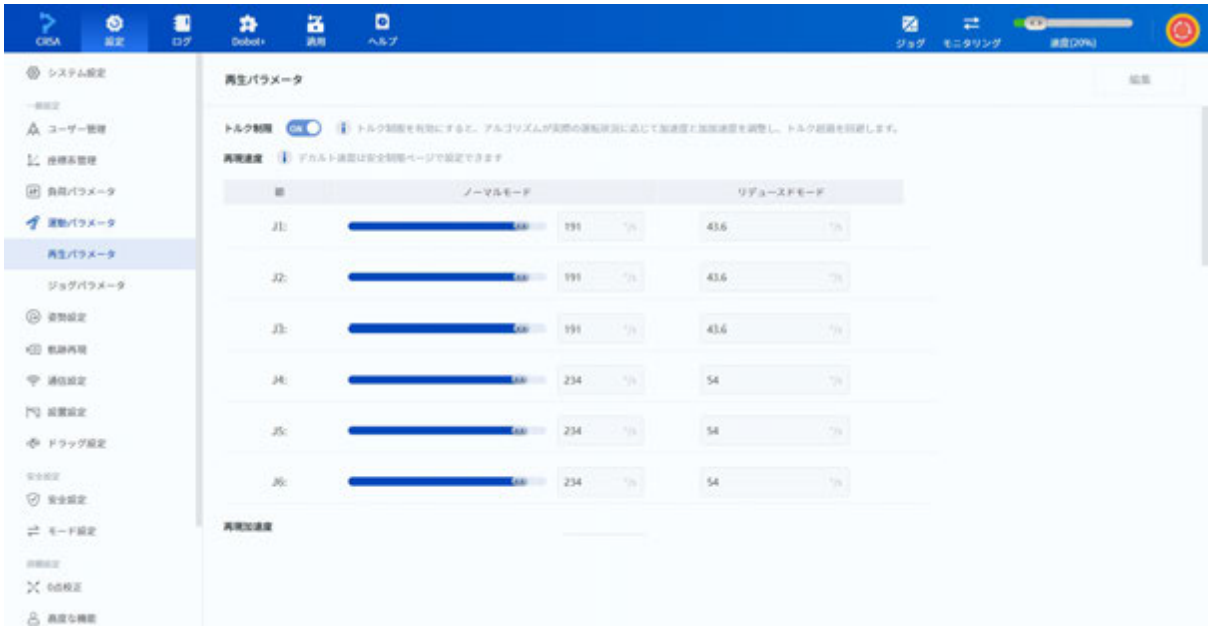
## 速度パラメータ

### 相対速度

オプションパラメータの `a` と `v` は、ロボットアームがこのモーションコマンドを実行する際の加速度と速度の比率を指定するために使用されます。

```
ロボット実際の運動速度 = 最大速度 x グローバル速度 x コマンド速度比率
ロボット実際の運動加速度 = 最大加速度 x コマンド速度比率
```

最大速度および加速度は**再現パラメータ**によって制御されます。これらはDobotStudio Proのモーションパラメータページで確認および変更できます。



グローバル速度は、DobotStudio Proの速度調整（上図右上隅）またはSpeedFactorコマンドで設定できます。

コマンド速度はモーションコマンドのオプションパラメータで指定される比率です。オプションパラメータでモーション加速度や速度の比率を指定しない場合、モーションパラメータで設定された値がデフォルトで使用されます（詳細はVelJ、AccJ、VelL、AccLコマンドを参照。コマンドで設定されていない場合のデフォルト値はすべて100です）。

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に運動
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に運動

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に直線運動

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に直線運動
```

**絶対速度**

直線および円弧運動コマンドのオプションパラメータ中のspeedは、ロボットがこの運動コマンド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、グローバルの速度比率から影響を受けないですが、**再現設定**の最大速度の制限を受けます（ロボットがリデュースドモードに入っている場合、減速後の最大速度の制限を受けます）。つまり、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:

```
MovL(P1,{speed = 1000})  -- 絶対速度1000mm/sでP1に直線運動
```

MovLでspeedが1000に設定されている場合、再現設定の最大速度2000より小さいため、ロボットアームは1000mm/sを目標速度として動作します。この目標速度は、現在のグローバル速度率には依存しません。しかし、ロボットアームが縮減モードにある場合（例えば縮減率が10%と仮定）、最大速度は200となり、1000より小さいため、ロボットアームは200mm/sを目標速度として動作します。

speedパラメータとvパラメータは相互排他的であり、両方が設定されている場合はspeedが優先されます。

## スムーズな遷移パラメータ

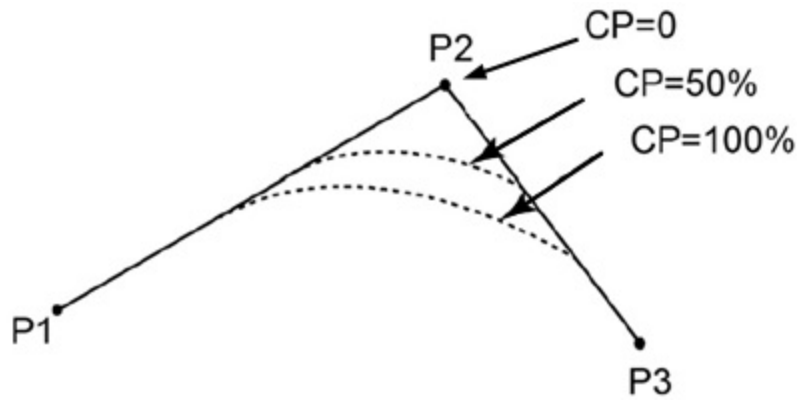
ロボットアームが複数のポイントを連続して移動する場合、スムーズな移行を行うことで中間ポイントを通過し、急な方向転換を避けることができます。ただし、ユーザーが指定した複数の経路ポイントが異なるツール座標系に基づいている場合、スムーズな移行は行えません。

オプションパラメータのcpまたはrは、現在のモーションコマンドから次のモーションコマンドへのスムーズな移行の比率（cp）またはスムーズな移行の半径（r）を指定します。**rが設定されている場合、cpパラメータは無視されます。**

> **説明:**
> 関節運動関連のコマンドはスムーズトランジション半径（r）の設定を対応していません。
> 各コマンドのオプションパラメータをご参照ください。

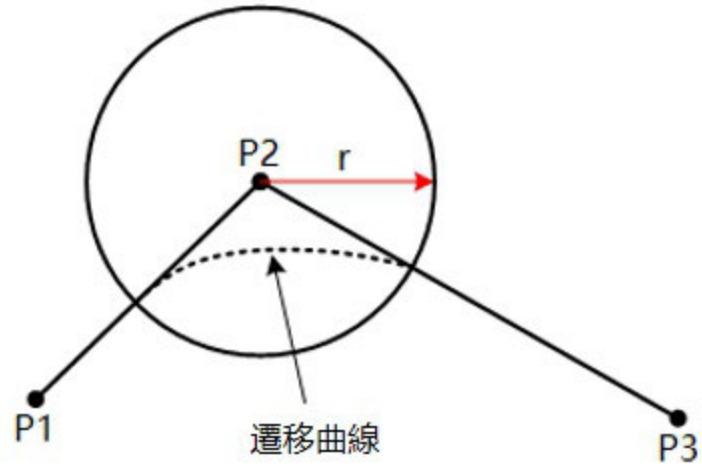
### CP

スムーズトランジション比率を設定する場合は、運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運動曲線は運動速度/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、運動速度/加速度が異なる場合の運動曲線の円弧は異なります。

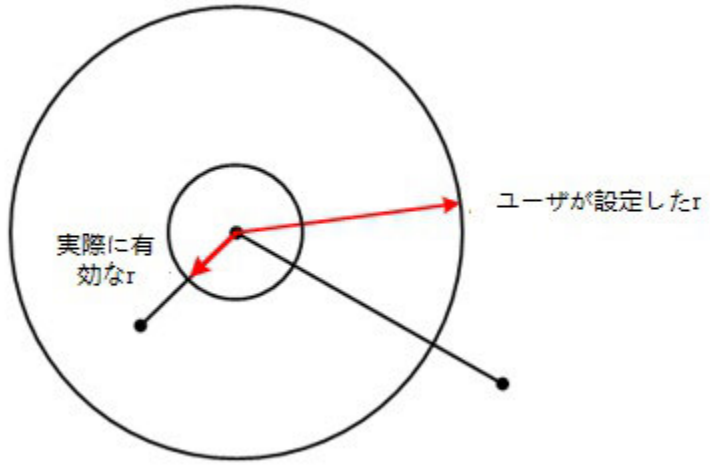



**R**

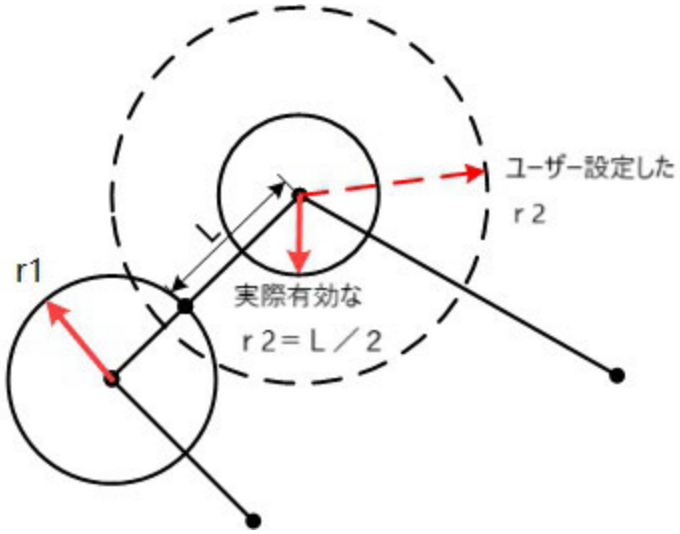
设置平滑过渡半径时，系统会以过渡点为圆心，根据指定半径计算过渡曲线。R过渡曲线不受运动速度/加速度影响，只由点位和过渡半径决定。




ユーザーが設定した移行半径が大きすぎる場合（開始点/終了点と移行点の間の距離を超える場合）、システムは自動的に開始点/終了点と移行点の間の短い距離の半分を移行半径として使用し、移行曲線を計算します。



連続する2つの移行半径（以下の図中のr1およびr2）が重なっている場合、システムは第1セグメントの移行が完了した点を第2セグメントの開始点として使用します。その後、移行半径が大きすぎる場合のロジックに従って、第2のrの実際の有効値を計算します。




**デフォルト値**

オプションパラメータでスムーズな移行の比率または半径を指定しない場合、モーションパラメータで設定されたスムーズな移行比率がデフォルトで使用されます（詳細はCPコマンドを参照。コマンドが設定されていない場合、デフォルト値は0です）。

**注意事項**

スムーズな移行を設定すると、ロボットアームの動作が中間ポイントを通過しない可能性があります。そのため、スムーズな移行が設定されている場合、2つのモーションコマンド間のIO信号出力や機能設定（例：安全スキンのオン/オフ）に関する指令は**移行中に実行されます。**

ロボットアームが正確に中間ポイントに到達した際に指令を実行したい場合、前のモーションコマンドのスムーズな移行パラメータを0に設定してください。

cpとrの実装方式の違いにより、スムーズな移行を必要とする2つのモーションコマンド間に他の動作に影響しない指令（例：条件判断、IO、機能設定）を挿入した場合、cpとrでは処理方法に違いがあります。

- **cp**方式を使用した場合、**ほとんどの指令は移行に影響しません。**ただし、指令の処理時間が長い場合（例：Wait指令）は例外です。

以下の例では、2つのモーションコマンドの間にif文を挿入しても、`cp` 方式のスムーズな移行には影響しません。

```
-- 正常に運動できます。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%で次の動作コマンドに遷移します。
MovL(P1,{cp=50})
if (DI(1)==ON)
then
    MovL(P2)
```

```
end
```

- r方式を使用した場合、**以下のホワイトリストに含まれる指令のみがスムーズな移行に影響を与えません。**その他の指令を挿入すると、r方式のスムーズな移行が無効になります。

```
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つの運動コマンドの間にifステートメントを挿入すると、rモードでのスムーズトランジション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、運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、ロボットは動作コマンド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、次のコマンドを直接実行します。

停止条件はLua構文を満たす任意の式を対応します。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未満の場合に運動を停止します
MovJ(P1,{stopcond="GetHoldRegs(id, 100, 1)[1] < 10"})
-- 変数var1が5に等しくない場合は運動を停止します
MovJ(P1,{stopcond="var1 ~= 5"})
```

**説明:**

- 停止条件を指定すると、スムーズな移行パラメータrは無効になります。
- 停止条件を指定した場合でも、スムーズな移行パラメータcpは有効ですが、移行セグメント（曲線部分）は停止条件の適用範囲外となります。
- 停止条件が指定された指令の前にスムーズな移行が設定されている場合でも、移行セグメントを離れてから停止条件がトリガーされます。

## 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。

# モーションコマンド

## コマンドリスト

モーションコマンドは、ロボットを制御して移動させ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。**

| コマンド | 機能 |
| --- | --- |
| MovJ | 関節動作 |
| MovL | 直線動作 |
| Arc | 円弧補間動作 |
| Circle | 全円の補間動作 |
| MovJIO | 関節動作とDOの出力 |
| MovLIO | 直直線動作とDOの出力 |
| GetPathStartPose | 軌跡の開始点を取得する |
| StartPath | 記録された軌跡を再現する |
| PositiveKin | 姿勢に対する関節角度の順解法 |
| InverseKin | 関節角度に対する姿勢の逆解法 |

## MovJ

**プロトタイプ:**

```
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：このコマンド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：このコマンドを実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。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、次のコマンドを実行します。

詳細は共通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点まで関節移動します。
MovJ(P1)
```

```
-- 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} })
```

```
-- ロボットは指定の姿勢まで関節移動します。姿勢はユーザー座標系1とツール座標系1に対応し、移動加速度と速度は共に50%、スムーズトランジション制御比率は50%です。
MovJ({pose={300,200,300,180,0,0} },{user=1,tool=1,a=50,v=50,cp=50})
```

```
-- 先に点を定義し、その後移動コマンドで呼び出します。実行結果は上記のコマンドと同じです。
customPoint={pose={300,200,300,180,0,0} }
MovJ(customPoint,{user=1,tool=1,a=50,v=50,cp=50})
```

```
-- 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：このコマンド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：このコマンド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：このコマンド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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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、次のコマンドを実行します。

詳細は共通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。姿勢はユーザー座標系1とツール座標系1に対応し、移動加速度と速度は共に50%、スムーズトランジション制御半径は5mmです。
MovL({pose={300,200,300,180,0,0} },{user=1,tool=1,a=50,v=50,r=5})
```

```
-- 先にポイントを定義し、その後移動コマンドで呼び出します。実行結果は上記のコマンドと同じです。
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

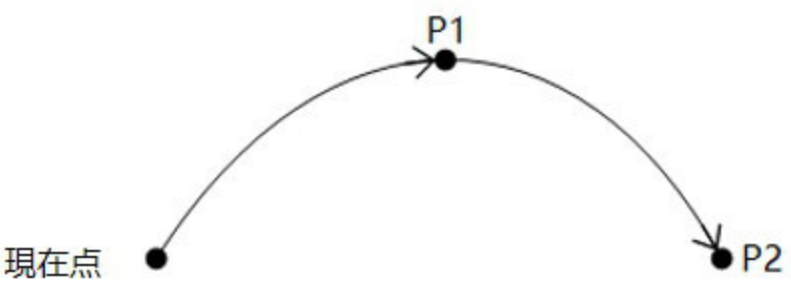
**プロトタイプ:**

```
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。

**オプションパラメータ:**

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：このコマンド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：このコマンドを実行する場合のロボットの移動速度です。値の範囲：(0,100]
- speed：このコマンドを実行する際のロボットの目標速度。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：スムーズトランジション制御の比率です。値の範囲：[0,100]
- r：スムーズトランジション制御半径です。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、次のコマンドを実行します。

詳細は共通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。ユーザー座標系とツール座標系はどちらも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

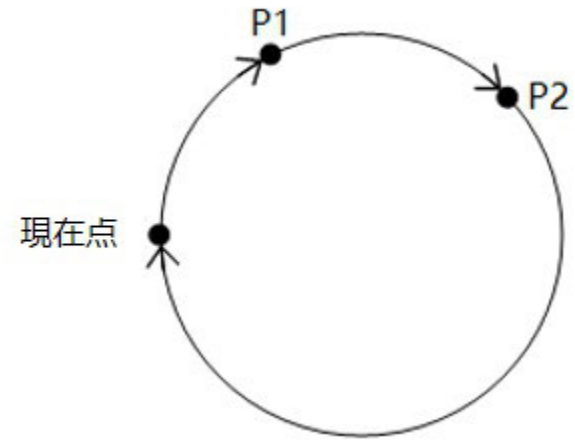
**プロトタイプ:**

```
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表します。値範囲は[1, 999]。

**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：このコマンド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：このコマンドを実行する場合のロボットの移動速度です。値の範囲：(0,100]
- speed：このコマンドを実行する際のロボットの目標速度。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：スムーズトランジション制御の比率です。値の範囲：[0,100]
- r：スムーズトランジション制御半径です。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、次のコマンドを実行します。

詳細は共通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。ユーザー座標系とツール座標系はどちらも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})
```

**可选参数:**

- user：目标点的用户坐标系。
- tool：目标点的工具坐标系。
- a：执行该条指令时的机械臂运动加速度比例。取值范围：(0,100]。
- v：执行该条指令时的机械臂运动速度比例。取值范围：(0,100]。
- cp：平滑过渡比例。取值范围：[0,100]。

平滑过渡会改变机械臂运动轨迹，对DO输出的时机造成影响，请谨慎使用。

详细介绍请参见通用说明。

**説明:**

現在位置から関節動作方式で目標点に移動し、移動中にデジタル出力ポートの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: トリガーモード。0は割合トリガー、1は距離トリガーを表します。システムは各関節角度を合成して角度ベクトルとし、終点と始点の角度差を移動の総距離として計算します。
  - Distance:

    指定する割合または角度。角度計算は合成角ベクトルを使用するため、割合モードを使用することを推奨します。その方が直感的です。

    - Distanceが0の場合、始点でトリガーします。
    - Distanceが正の場合、始点からの割合または角度を指定します。
    - Distanceが負の場合、目標点からの割合または角度を指定します。
    - Modeが0の場合、Distanceは総角度の割合を表します。範囲：(0,100]。
    - Modeが1の場合、Distanceは角度の値を表します。単位：°。
  - Index: DO端子の番号。
  - Status: 設定するDOの状態。0およびOFFは信号なし、1およびONは信号ありを表します。

**オプションパラメータ:**

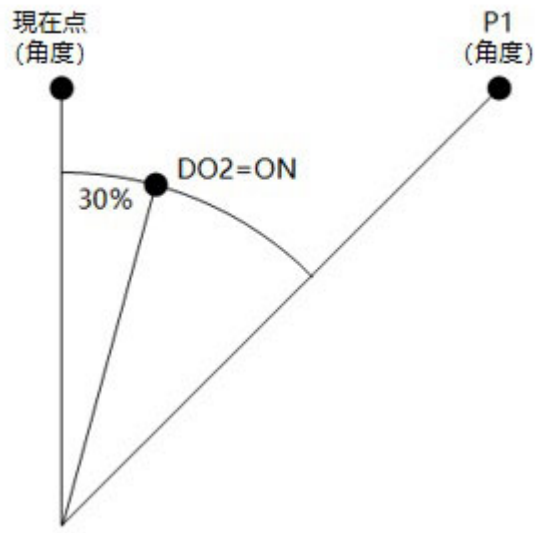
- user: 目標点のユーザー座標系。
- tool: 目標点のツール座標系。
- a: この指令実行時のロボットアームの加速度比率。範囲：(0,100]。
- v: この指令実行時のロボットアームの速度比率。範囲：(0,100]。
- cp: スムーズな移行比率。範囲：[0,100]。

スムーズな移行はロボットアームの動作軌跡を変える可能性があり、DO出力のタイミングに影響を与えるため、慎重に使用してください。

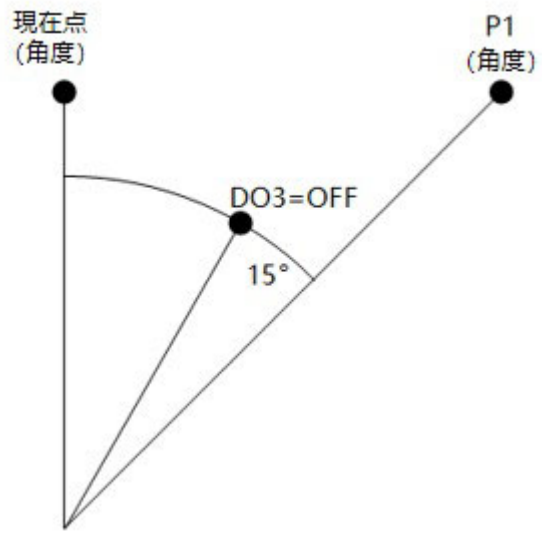
詳細は共通説明をご参照ください。

**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} })
```




```
-- 初期設定ではマニピュレータがP1点ジョイントに移動します。始点から30%の位置に移動する場合はDO2をオンに
```

```
設定し、終点から20%の位置に移動する時はDO2をオフに設定します。
MovJIO(P1, { {0, 30, 2, 1}, {0, -20, 2, 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:**

現在位置から直線動作方式で目標点に移動し、移動中にデジタル出力ポートの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割合に達した際に、指定されたDO（デジタル出力）をトリガーします。複数設定可能で、各セットには以下のパラメータが含まれます:

  - Mode: トリガーモード。0は割合トリガー、1は距離トリガーを表します。
  - Distance:

    指定する割合または距離。

    - Distanceが0の場合、始点でトリガーします。
    - Distanceが正の場合、始点からの割合または距離を指定します。
    - Distanceが負の場合、目標点からの割合または距離を指定します。
    - Modeが0の場合、Distanceは総距離の割合を表します。範囲：(0,100]。
    - Modeが1の場合、Distanceは距離の値を表します。単位：mm。
  - Index: DO端子の番号。
  - Status: 設定するDOの状態。0およびOFFは信号なし、1およびONは信号ありを表します。

**オプションパラメータ:**

- user: 目標点のユーザー座標系。
- tool: 目標点のツール座標系。
- a: この指令実行時のロボットアームの加速度比率。範囲：(0,100]。
- v: この指令実行時のロボットアームの速度比率。範囲：(0,100]。

- speed: この指令実行時のロボットアームの目標速度。範囲：[1, 最大移動速度]、単位：mm/s。

  このパラメータを設定した場合、**v**パラメータは無視されます。

- cp: スムーズな移行比率。範囲：[0,100]。
- r: スムーズな移行半径。範囲：[0,100]、単位：mm。

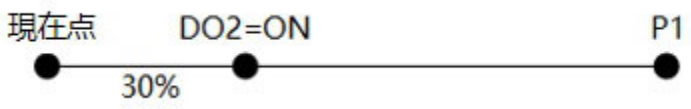
  このパラメータを設定した場合、cpパラメータは無視されます。

スムーズな移行はロボットアームの動作軌跡を変える可能性があり、DO出力のタイミングに影響を与えるため、慎重に使用してください。

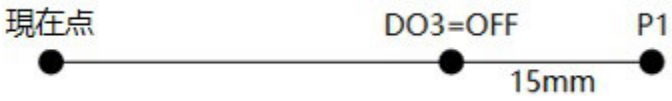
詳細は共通説明をご参照ください。

**例:**

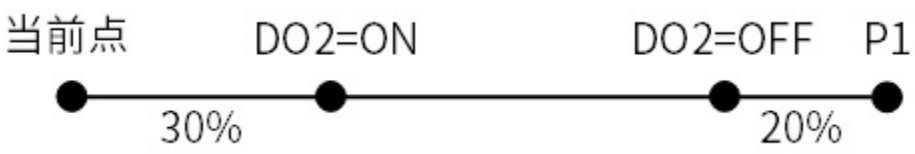
```
-- ロボットは、デフォルト設定ではP1点に向けて直線移動します。開始点から30%の位置に移動すると、DO2がオンになるように設定します。
MovLIO(P1, { {0, 30, 2, 1} })
```




```
-- 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} })
```




```
-- 初期設定ではロボットアームは点P1まで直線移動します。始点から30%の位置に移動する場合はDO2をオンにします。終点から20%の位置に移動する時はDO2をオフに設定します。
MovLIO(P1, { {0, 30, 2, 1}, {0, -20, 2, 0} })
```




## GetPathStartPose

**プロトタイプ:**

```
GetPathStartPose(string)
```

**説明:**

通常、StartPath指令と組み合わせて使用され、このコマンドを呼び出して軌跡再現の開始点に移動します。軌跡ファイルによって、ポイントデータのタイプ（ティーチポイント/関節/姿勢）が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（拡張子を含む）。

**例:**

```
-- 軌跡ファイル「track.csv」の開始点を取得し、表示します。
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}}
```

## 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。このコマンドを呼び出す前に、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は、ユーザーがDobotStudio Proで記録した軌跡ファイルです。
-- 軌跡ファイルの開始点に関節動作で移動した後、2倍の速度でファイル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は、ユーザーがDobotStudio Proで記録した軌跡ファイルです。
--  track.csvは、ユーザーがDobotStudio Proで記録した軌跡ファイルです。
local StartPoint = GetPathStartPose("track.csv")
MovJ(StartPoint)
StartPath("track.csv", {isConst = 1})
```

```
-- customtrack.csvは、ユーザーが自分で作成した軌跡ファイルで、サンプリング間隔は20msです。
-- 軌跡ファイルの開始点に関節動作で移動した後、一定速度でファイルに記録された軌跡を再現します。フィルタ係数は1（記録された軌跡を完全に再現）、ユーザー座標系とツール座標系はともに0とします。
local StartPoint = GetPathStartPose("customtrack.csv")
MovJ(StartPoint)
StartPath("customtrack.csv", {isConst = 1, sample = 20, freq = 1, user = 0, tool = 0})
```

# 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。

**戻り値**

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

**例:**

```
-- 関節座標は {0,0,-90,0,90,0} であり、グローバル ユーザー/ツール座標系におけるロボット アームの端のデカルト座標が計算されます。
PositiveKin({joint={0,0,-90,0,90,0} })
```

```
-- 関節座標は {0,0,-90,0,90,0} であり、グローバル ユーザー/ツール座標系におけるロボット アームの端のデカルト座標が計算されます。
PositiveKin({joint={0,0,-90,0,90,0} },{user=1,tool=1})
```

# 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。

**戻り値**

- 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:**

```
-- グローバルユーザー/ツール座標系におけるロボットの末端の直交座標は {300, 200, 300, 180, 0, 0} として関節座標を計算し、最も近い解を取得します。
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} })
```

```
-- ユーザー座標系1およびツール座標系1におけるロボットの末端の直交座標は {300, 200, 300, 180, 0, 0} として関節座標を計算し、最も近い解を取得します。
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} }, {user=1, tool=1})
```

```
-- グローバルユーザー/ツール座標系におけるロボットの末端のデカルト座標は {300, 200, 300, 180, 0, 0} として関節座標を計算し、関節角度 {90, 30, -90, 180, 30, 0} の最も近い解を選択します。
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} }, {useJointNear = true, jointNear = { joint = {90, 30, -90, 180, 30, 0} } })
```

# 相対運動コマンド

## コマンドリスト

相対移動コマンド関数はロボット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。**

| コマンド | 機能 |
| --- | --- |
| RelPointUser | 点をユーザー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 | 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：偏移する指定点。
- {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。

**戻り値:**

- 指定点がティーチングポイントの場合は、偏移後のティーチングポイントの定数を返します。
- 指定点がジョイント変数または姿勢変数の場合、偏移後の姿勢変数を返します: {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：偏移する指定点。
- {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。

**戻り値:**

オフセット後の姿勢変数：`{"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}：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：目標点の工具座標系。
- a：このコマンド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：このコマンドを実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。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、次のコマンドを実行します。

詳細は共通説明をご参照ください。

**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})
```

オプションのパラメーターに関連するその他の例については、MovJ を参照してください。

## 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：目標点の工具座標系。
- a：このコマンド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：このコマンド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：このコマンド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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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、次のコマンドを実行します。

詳細は共通説明をご参照ください。

**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})
```

オプションのパラメーターに関連するその他の例については、MovJ を参照してください。

## 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：目標点の工具座標系。
- a：このコマンド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：このコマンドを実行する場合のロボットの移動速度。値の範囲：(0,100]

- cp：スムーズトランジション制御の比率。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、次のコマンドを実行します。

詳細は共通説明をご参照ください。

**例:**

```
-- ロボットはデフォルト設定により、グローバルユーザー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})
```

オプションのパラメーターに関連するその他の例については、MovJ を参照してください。

## 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：目標点の工具座標系。
- a：このコマンド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：このコマンド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：このコマンド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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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、次のコマンドを実行します。

詳細は共通説明をご参照ください。

**例:**

```
-- ロボットはデフォルト設定により、グローバルユーザー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})
```

オプションのパラメーターに関連するその他の例については、MovJ を参照してください。

## 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：このコマンド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：このコマンド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]

詳細は共通説明をご参照ください。

**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：偏移する指定点。
- 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°

**戻り値:**

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)
```

# モーションパラメータ

## コマンドリスト

モーションパラメータ関数は、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されます。**使用する前に共通説明を読んでください。**

| コマンド | 機能 |
| --- | --- |
| CP | スムーズトランジション制御の比率の設定 |
| 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 | ロボットのリアルタイム姿勢の取得 |
| GetAngle | ロボット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 | 関節移動コマンドの実行可能性の確認 |
| CheckMovL | 直線または円弧移動コマンドの実行可能性の確認 |
| SetSafeWallEnable | 指定した安全壁のオンまたはオフ |
| SetWorkZoneEnable | 指定の安全エリアのオン・オフ |
| SetCollisionLevel | 衝突レベルの設定 |
| SetBackDistance | 衝突戻り距離の設定 |

# CP

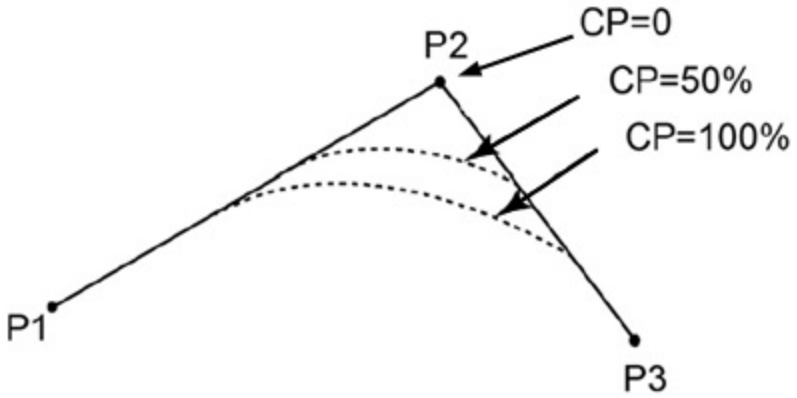
**プロトタイプ:**

```
CP(R)
```

**説明:**

スムーズトランジション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スムーズトランジション制御の比率は、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：スムーズトランジション制御の比率。値の範囲：[0, 100]

**例:**

```
-- ロボットアームは現在P1点に位置しています。50%の平滑な移行率でP2点に向けて曲線的に移動し、最終的にP3点に到達します。
CP(50)
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]

**示例:**

```
-- ロボットは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:**

ロボットの全体的な移動速度を設定します。詳細はモーションコマンド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:**

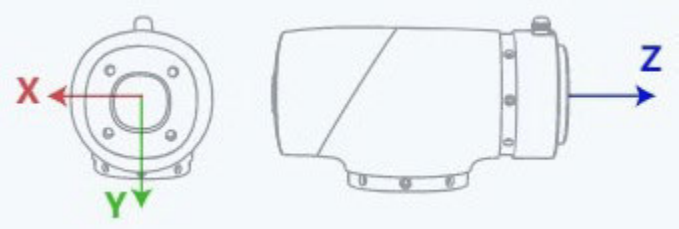
```
SetPayload(payload, {x, y, z})
SetPayload(name)
```

**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:**

```
-- 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。ユーザーがその他コマンドを呼び出す場合、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)
-- 以下の例では、グローバルユーザー座標系の適用範囲を説明します。
MovL({pose={300,200,300,180,0,0} }) -- グローバルユーザー座標系（1）を使用
MovL({pose={300,200,300,180,0,0} },{user=2}) -- オプションパラメータで
MovL(P1) -- P1はティーチングポイントで、そのティーチングポイントに付随するユーザー座標系を使用
```

## SetUser

**プロトタイプ:**

```
SetUser(index,table,type)
```

**説明:**

指定されたユーザー座標系を変更します。

**必須パラメータ:**

- **index**: ユーザー座標系のインデックス。このインデックスは、制御ソフトウェアであらかじめ追加された座標系である必要があります。指定された座標系インデックスが存在しない場合、プログラムの実行が停止し、エラーが発生します。
- **table**: 変更後のユーザー座標系を[x, y, z, rx, ry, rz]の形式で指定します。rx、ry、rzの回転値が含まれる場合、CalcUserコマンドを使用して値を取得することを推奨します。

**オプションパラメータ:**

- type: グローバル保存するかどうか。デフォルト値は0。
  - **0**: このコマンドで変更された座標系は、現在のプログラム実行中のみ有効で、プログラム停止後は元の値に戻ります。
- **1**: このコマンドで変更された座標系はグローバルに保存され、プログラム停止後も変更後の値が保持されます。ただし、`type=1`（グローバル保存）の場合、ユーザーは手動で対応する座標系設定画面に切り替え、再度戻ることで更新後の値を確認する必要があります（手動で画面をリフレッシュする必要があります）。

**例:**

```
-- ユーザー座標系1をCalcUserで計算した新しい座標系に変更します。この変更はプロジェクト実行中のみ有効です。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
SetUser(1,newUser)
User(1) -- グローバルユーザー座標系を1に切り替え
```

```
-- ユーザー座標系1をCalcUserで計算した新しい座標系に変更し、グローバルに保存します。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
SetUser(1,newUser,1)
MovL(P1,{user=1}) -- オプションパラメータを使用して、このコマンドの参照ユーザー座標系を1に指定
```

# 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。

**戻り値:**

計算された[x、y、z、rx、ry、rz]形式のユーザー座標系です。

**例:**

```
-- 以下の計算プロセスは次のように説明できます：初期姿勢がユーザー座標系1と同じ座標系を基準座標系に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転させた結果、新しい座標系newUserが得られます。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
```

```
--  以下の計算プロセスは次のように説明できます：初期姿勢がユーザー座標系1と同じ座標系を、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}だけ回転させた結果、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。ユーザーがその他コマンドを呼び出す場合、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)
-- 以下の例では、グローバルツール座標系の適用範囲を説明します。
MovL({pose={300,200,300,180,0,0} }) -- グローバルツール座標系（1）を使用
MovL({pose={300,200,300,180,0,0} },{tool=2}) -- オプションパラメータで指定したツール座標系（2）を使用

MovL(P1) -- P1はティーチングポイントで、そのティーチングポイントに付随するツール座標系を使用
```

# SetTool

**プロトタイプ:**

```
SetTool(index,table,type)
```

**説明:**

指定されたツール座標系を変更します。

**必須パラメータ:**

- **index**: ツール座標系のインデックス。このインデックスは、制御ソフトウェアであらかじ

め追加された座標系である必要があります。指定された座標系インデックスが存在しない場合、プログラムの実行が停止し、エラーが発生します。
- **table**: 変更後のツール座標系を[x, y, z, rx, ry, rz]の形式で指定します。rx、ry、rzの回転値が含まれる場合、CalcToolコマンドを使用して値を取得することを推奨します。

**オプションパラメータ:**

- type: グローバル保存するかどうか。デフォルト値は0。
  - **0**: このコマンドで変更された座標系は、現在のプログラム実行中のみ有効で、プログラム停止後は元の値に戻ります。
- **1**: このコマンドで変更された座標系はグローバルに保存され、プログラム停止後も変更後の値が保持されます。ただし、`type=1`（グローバル保存）の場合、ユーザーは手動で対応する座標系設定画面に切り替え、再度戻ることで更新後の値を確認する必要があります（手動で画面をリフレッシュする必要があります）。

**例:**

```
-- ツール座標系1をCalcToolで計算した新しい座標系に変更します。この変更はプロジェクト実行中のみ有効です。

newTool = CalcTool(1,1,{10,10,10,10,10,10})
SetTool(1,newTool)
Tool(1) -- グローバルツール座標系を1に切り替え
```

```
-- ツール座標系1をCalcToolで計算した新しい座標系に変更し、グローバルに適用します。
newTool = CalcTool(1,1,{10,10,10,10,10,10})
SetTool(1,newTool,1)
MovL(P1,{tool=1}) -- オプションパラメータを使用して、このコマンドの参照ツール座標系を1に指定
```

# 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。

**戻り値:**

計算された[x、y、z、rx、ry、rz]形式のツール座標系です。

**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:**

ロボットのリアルタイムの姿勢を取得します。

2つの移動コマンド間でこのコマンドを呼び出す場合、得られるポイントはスムーズトランジション制御設定の影響を受けます。

- スムーズトランジション制御機能（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。
- スムーズトランジション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。

**戻り値:**

ロボットの現在の姿勢：{pose：[x、y、z、rx、ry、rz]}

**例:**

```
-- ロボットアームがまず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)
```

```
-- ロボットアームが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() --获取P1的位姿
MovJ(P2)
```

```
-- 現在の姿勢をユーザー座標系1およびツール座標系1の下で取得します。
local currentPose = GetPose(1,1)
```

# 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つの移動コマンド間でこのコマンドを呼び出す場合、得られる間接角度はスムーズトランジション制御設定の影響を受ける場合があります。

- スムーズトランジション制御機能（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。
- スムーズトランジション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。

**戻り値:**

ロボットの現在の関節角度:{joint：[j1、j2、j3、j4、j5、j6]}

**例:**

```
-- ロボットアームがまずP1に移動し、その後現在の関節角度に戻ります。
local currentAngle = GetAngle()
MovJ(P1)
MovJ(currentAngle)
```

```
-- ロボットアームが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
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エンコーダの現在位置を取得し、変数abzに代入します。
local abz = GetABZ()
```

# CheckMovJ

**プロトタイプ:**

```
CheckMovJ(P, {user = 1, tool = 0, a = 20, v = 50, cp = 100})
```

**説明:**

現在の位置から目標点へのジョイントモーションの実行可能性を確認します。システムは移動軌跡全体を計算し、軌跡内に到達不可能なポイントがないかを検査します。

**必須パラメータ:**

- **P**: 目標点。

**オプションパラメータ:**

- **user**: 目標点のユーザー座標系。
- **tool**: 目標点のツール座標系。
- **a**: このコマンドを実行する際のロボットの加速度比率。取値範囲：(0,100]。
- **v**: このコマンドを実行する際のロボットの速度比率。取値範囲：(0,100]。
- **cp**: スムーズな移行の比率。取値範囲：[0,100]。

詳細は一般的な説明を参照してください。

**戻り値:**

検査結果:

- 0: エラーなし
- 16: 終点が肩の特異点に近い
- 17: 終点で逆計算で解なし
- 18: 終点が逆計算で範囲外
- 22: 姿勢切替エラー
- 26: 終点が手首の特異点に近い
- 27: 終点が肘の特異点に近い
- 29: 速度パラメータエラー
- 30: 全てのパラメータの逆計算が失敗
- 32: 軌跡に肩の特異点がある
- 33: 軌跡に逆計算が存在しないポイントがある
- 34: 軌跡に逆計算範囲外のポイントがある
- 35: 軌跡に手首の特異点がある
- 36: 軌跡に軸の特異点がある
- 37: 軌跡にジョイントのジャンプポイントがある

**例:**

```
-- 関節動作のデフォルト設定でP1に到達可能かを確認し、到達可能であれば関節動作でP1に移動します。
local status=CheckMovJ(P1)
if(status==0)
then
    MovJ(P1)
    status=CheckMovJ(P2) -- P1に到達後、P1からP2への可達性をチェック
    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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**戻り値:**

検査結果。

- 0: エラーなし
- 16: 終点が肩の特異点に近い
- 17: 終点で逆計算で解なし
- 18: 終点が逆計算で範囲外
- 22: 姿勢切替エラー
- 26: 終点が手首の特異点に近い
- 27: 終点が肘の特異点に近い
- 29: 速度パラメータエラー
- 30: 全てのパラメータの逆計算が失敗
- 32: 軌跡に肩の特異点がある
- 33: 軌跡に逆計算が存在しないポイントがある
- 34: 軌跡に逆計算範囲外のポイントがある
- 35: 軌跡に手首の特異点がある
- 36: 軌跡に軸の特異点がある
- 37: 軌跡にジョイントのジャンプポイントがある

**例:**

```
--直線動作のデフォルト設定でP1に到達可能かを確認し、到達可能であれば直線動作でP1に移動します。
local status=CheckMovL(P1)
if(status==0)
then
    MovL(P1)
    status=CheckMovL(P2) -- P1に到達後、P1からP2への可達性をチェック
```

```
    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オンまたはオフにします。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：安全壁をオンにします。
  - False：安全壁をオフにします。

**例:**

```
-- インデックス1のセーフティウォールを有効にします。
SetSafeWallEnable(1,true)
```

## SetWorkZoneEnable

**プロトタイプ:**

```
SetWorkZoneEnable(index,value)
```

**説明:**

指定した安全エリア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安全エリアの状態は、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：設定する安全エリアのインデックス。事前に制御ソフトウェアで対応する安全エリアを追加する必要があります。値の範囲：[1,6]

- value：
  - True：安全エリアをオンにします。
  - False：安全エリアをオフにします。

**例:**

```
-- インデックス1の安全エリアを開きます。
SetWorkZoneEnable(1,true)
```

## SetCollisionLevel

**プロトタイプ:**

```
SetCollisionLevel(level)
```

**説明:**

衝突検知レベルを設定します。CRAモデルの場合、このコマンドは安全制限のTCP力と電力値を設定します。このコマンドで設定された値は、現在のプロジェクト実行中のみ有効であり、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: 衝突検知レベル。0は衝突検知を無効化、1～5は数字が大きい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

## コマンドリスト

IOコマンドは、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。

| コマンド | 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。

**戻り値:**

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。

**戻り値:**

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、コマンド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コマンドが実行されます。

**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。

**戻り値**

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。

**戻り値:**

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。

**戻り値:**

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。

**戻り値**

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)
```

# 末端ツール

## コマンドリスト

末端ツールコマンドは、ロボットシステムの末端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されます。

| コマンド | 機能 |
| --- | --- |
| ToolDI | 末端デジタル入力ポートの状態を読み取ります |
| ToolDO | 末端デジタル出力ポートの状態を設定します(シーケンスコマンド) |
| GetToolDO | 末端デジタル出力ポートの現在の状態を取得します |
| ToolAI | 末端アナログ入力ポートの値を読み取ります |
| SetToolMode | 末端端子多重化通信モードを設定します |
| SetToolPower | 末端ツールの電源状態を設定します |
| SetTool485 | 末端ツールの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:**

末端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：末端DI端子の番号。

**戻り値:**

対応するDI端子の状態（ON/OFF）。

**例:**

```
-- 末端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:**

末端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：末端DO端子の番号。
- ON|OFF：設定するDOポートの状態。

**例:**

```
-- 末端DO1をONに設定します。
ToolDO(1,ON)
```

## GetToolDO

**プロトタイプ:**

```
GetToolDO(index)
```

**説明:**

末端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：末端DO端子の番号。

**戻り値**

対応する末端DO端子の状態（ON/OFF）。

**例:**

```
-- 末端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:**

末端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：末端AI端子の番号。

**戻り値:**

対応するAI端子の値。

**例:**

```
-- 末端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。

  **値の範囲:**
  - 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

  **例:**
  - 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：ロボットに複数の末端エンドコネクタがあるとき、エンドコネクタの指定に使用します。

- 1はエンドコネクタ1を表します。
- 2はエンドコネクタ2を表します。

**例:**

```
-- 終端多重端末を485モードに設定します。
SetToolMode(1,0)
```

```
-- CR20A末端エンドコネクタ2の多重端子を485モードに設定します。
SetToolMode(1,0,2)
```

```
-- 末端多重化端子をアナログ入力モードに設定します。両チャンネルとも0～10Vの電圧入力モードになります。
SetToolMode(2,0)
```

```
-- 末端多重化端子をアナログ入力モードに設定します。AI1は0～10V 電圧入力モード、AI2は電流モードです。。
SetToolMode(2,10)
```

## GetToolMode

**プロトタイプ:**

```
GetToolMode(identify)
```

**説明:**

終端多重端子の現在のモードを取得します。

**選択可能なパラメータ:**

identify：ロボットに複数の終端多重端子がある場合、終端多重端子を指定するために使用されます。デフォルト値は 1 です。

- 1はエンドコネクタ1を表します。
- 2はエンドコネクタ2を表します。

**戻り値:**

- mode：多重化端末モード。
- type：アナログ入力モード。

詳細については、SetToolMode の同名のパラメータを参照してください。

**例:**

```
-- 末端多重化端子のモードを取得します。
local mode,type = GetToolMode()
```

```
-- CR20A末端エンドコネクタ2の多重端子のモードを取得します。
local mode,type = GetToolMode(2)
```

## SetToolPower

**プロトタイプ:**

```
SetToolPower(status)
```

**説明:**

末端エフェクタの電源供給状態を設定します。通常、末端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、末端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:**
端末の電源をOFFにすると、DO端子も無効になります。

**必須パラメータ:**

status：末端ツールの電源状態。

- 0：電源オフ
- 1：電源オン

**例:**

```
-- 末端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:**

末端ツールの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
  - 「E」は偶数パリティ
  - 「N」はパリティビットなし
  - 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0.5，1，1.5，2です。デフォルト値は1です。

誤差が ±0.1 以内の場合、最も近い位置が自動的に選択されます (0.4001 は値 0.5 をとり、0.3999 はエラーを報告し、1.09999 は値 1 をとります)。

- identify：ロボットに複数の末端エンドコネクタがあるとき、エンドコネクタの指定に使用します。1はエンドコネクタ1を表し、2はエンドコネクタ2を表します。

**例:**

```
-- 末端ツールの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

## コマンドリスト

TCP&UDP関数は、TCPやUDP通信を行うために使用されます。

| コマンド | 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

**戻り値:**

- 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。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。

**戻り値:**

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。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。

**戻り値:**

- 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、タイムアウトなし、受信データはtable形式で保存されます。
-- socketは、TCPCreateによって正常に返されたソケットオブジェクトです。
err, RecBuf = TCPRead(socket) -- RecBufのデータ型はtableです
```

```
-- TCPデータを受信、タイムアウトは5秒、受信データはtable形式で保存されます。
-- socketは、TCPCreateによって正常に返されたソケットオブジェクトです。
err, RecBuf = TCPRead(socket,5) -- RecBufのデータ型はtableです
```

```
-- TCPデータを受信、タイムアウトなし、受信データは文字列形式で保存されます。
-- socketは、TCPCreateによって正常に返されたソケットオブジェクトです。
err, RecBuf = TCPRead(socket,0,"string") -- RecBufのデータ型は文字列です
```

# 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。

**戻り値：**

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」、タイムアウトなし。
```

```
TCPWrite(socket, "test") -- socketは、TCPCreateによって正常に返されたソケットオブジェクトです。
```

```
-- TCPデータを送信します。データコンテンツは「test」、タイムアウトは5秒。
TCPWrite(socket, "test", 5) -- socketは、TCPCreateによって正常に返されたソケットオブジェクトです。
```

# 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。

**戻り値:**

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：サーバーを作成するかどうか。
  - true：サーバーの作成を示します。
  - false：クライアントの作成を示します。

- IP：サーバーとクライアントを作成するときに、ピアの IP アドレスを入力します。これはロボット アームの IP アドレスと同じネットワーク セグメント内にある必要があり、競合しません。
- port：
  - サーバーを作成するときは、ローカル側と反対側の両方がこのポートを使用することを意味します。すでにシステムが占有しているポートは使用しないでください。詳細については、TCPCreate のパラメータの説明を参照してください。。
  - クライアントを作成する場合、ピアのポートを示します。このとき、ローカル エンドはデータを送信するときにランダムなポートを使用します。。

**戻り値:**

- 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1:**

```
-- UDPサーバーを作成します。
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)
```

**例2:**

```
-- UDP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。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。

**戻り値:**

- 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。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。

**戻り値:**

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

## コマンドリスト

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。

| コマンド | 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 | 単一書き込み | 連続書き込み |
| --- | --- | --- | --- |
| 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:**

TCP/IPをもとに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。詳細は付録A Modbusデー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。

**オプションパラメータ：**

- slave_id：スレーブID。
- isRTU：ブール値。
  - isRTUがfalseの場合、コントローラーのネットワークポートによるModbus TCP通信を確立します。
  - isRTUがtrueの場合、ロボット末端のRS485によるModbus RTU通信を確立します。ポート60000のみが使用可能です。

> ⚠ **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。

**戻り値：**

- 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コマンド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：**

```
-- ModbusTCPマスタを作成し、ロボット内蔵のスレーブと接続します。
-- IPはロボットのIPで、ポートは502、スレーブIDは指定しません。
local ip = "192.168.5.1"
local port = 502
local err = 0
local id = 0
err, id = ModbusCreate(ip, port)
```

```
-- ModbusTCPマスタを作成し、指定したスレーブと接続します。
-- スレーブのIPは192.168.5.123、ポートは503、スレーブIDは1
local ip = "192.168.5.123"
local port = 50
local err = 0
local id = 0
err, id = ModbusCreate(ip, port, 1)
```

```
-- ModbusRTU-over-TCPマスタを作成し、指定したスレーブと接続します。
-- スレーブのIPは192.168.5.123、ポートは503、スレーブIDは1。
local ip = "192.168.5.123"
local port = 503
local err = 0
local id = 0
err, id = ModbusCreate(ip, port, 1, true)
```

## ModbusRTUCreate

**プロトタイプ:**

```
ModbusRTUCreate(slave_id, baud, parity, data_bit, stop_bit)
```

**説明:**

コントローラーのRS485インターフェースによるModbusマスターを作成し、スレーブとの接続を確立します。同時に最大で15のデバイスとの接続を対応します。

**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。

**オプションパラメータ:**

- parity：パリティビットかどうか。
  - 「O」：奇数パリティ
  - 「E」：偶数パリティ
  - 「N」：パリティビットなし

    - 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。

**戻り値:**

- 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、後でその他Modbusコマンドを呼び出すときに使用します。

**例:**

```
-- Modbusマスタを作成し、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。
err, id = ModbusRTUCreate(1, 115200)
```

```
-- Modbusマスタを作成し、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、パリティなし（N）、データビット長は8、ストップビット長は2。
err, id = ModbusRTUCreate(1, 115200, "N", 8, 2)
```

## 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。

**戻り値:**

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スレーブの接点レジスタアドレスの値を読み取ります。対応するModbus機能コードは02です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 接点レジスタの開始アドレス。
- **count**: 連続して読み取る接点レジスタの数。取値範囲は [1, 2000] です（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**戻り値:**

接点レジスタアドレスの値がtable形式で返されます。tableの最初の値は接点レジスタの開始アドレスに対応する値です。

**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:**

指定されたデータ型に従い、Modbusスレーブの入力レジスタアドレスの値を読み取ります。対応するModbus機能コードは04です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 入力レジスタの開始アドレス。
- **count**: 連続して読み取る入力レジスタの数。取値範囲は [1, 125]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**オプションパラメータ:**

- type: データ型。
    - 空の場合、デフォルトでU16。
    - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
    - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
    - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
    - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**戻り値:**

入力レジスタアドレスの値がtable形式で返されます。tableの最初の値は入力レジスタの開始アドレスに対応する値です。

**例:**

```
-- アドレス2048から16ビットの符号なし整数を1つ読み取ります。
data = GetInRegs(id, 2048, 1)
```

```
-- 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")
```

```
-- アドレス2048から32ビット単精度浮動小数点数を2つ読み取ります。
data = GetInRegs(id, 2048, 4, "F32")
```

# GetCoils

**プロトタイプ:**

```
GetCoils(id, addr, count)
```

**説明:**

Modbusスレーブのコイルレジスタアドレスの値を読み取ります。対応するModbus機能コードは01です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: コイルレジスタの開始アドレス。
- **count**: 連続して読み取るコイルレジスタの数。取値範囲は [1, 2000]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**戻り値:**

コイルレジスタアドレスの値がtable形式で返されます。tableの最初の値はコイルレジスタの開始アドレスに対応する値です。

**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:**

指定された値をコイルレジスタの指定アドレスに書き込みます。対応するModbus機能コードは05（単一書き込み）および0F（複数書き込み）です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: コイルレジスタの開始アドレス。
- **count**: 連続して読み取るコイルレジスタの数。取値範囲は [1, 1968]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**戻り値:**

戻り値は0または-1です。0は設定成功を示し、-1は設定失敗を示します。

**例:**

```
-- アドレス1024から連続して5つのコイルに値を書き込みます。
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:**

指定されたデータ型に従い、Modbusスレーブの保持レジスタアドレスの値を読み取ります。対応するModbus機能コードは03です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 保持レジスタの開始アドレス。
- **count**: 連続して読み取る保持レジスタの数。取値範囲は [1, 125]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**オプションパラメータ:**

- type: データ型。
  - 空の場合、デフォルトでU16。
  - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
  - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
  - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
  - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**戻り値:**

保持レジスタアドレスの値がtable形式で返されます。tableの最初の値は保持レジスタの開始アドレスに対応する値です。

**例:**

```
-- アドレス2048から16ビットの符号なし整数を1つ読み取ります。
data = GetHoldRegs(id, 2048, 1)
```

```
-- 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")
```

```
-- アドレス2048から32ビット単精度浮動小数点数を2つ読み取ります。
data = GetHoldRegs(id, 2048, 4, "F32")
```

# SetHoldRegs

**プロトタイプ:**

```
SetHoldRegs(id, addr, count, table, type)
```

**説明:**

指定されたデータ型に従い、指定した値を保持レジスタの指定アドレスに書き込みます。対応するModbus機能コードは06（単一書き込み）および10（複数書き込み）です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 保持レジスタの開始アドレス。
- **count**: 連続して保持レジスタに書き込む数。取値範囲は [1, 123]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。
- **table**: 保持レジスタに書き込む値がtable形式で返されます。tableの最初の値は保持レジスタの開始アドレスに対応します。

**オプションパラメータ:**

- type: データ型。
  - 空の場合、デフォルトでU16。
  - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
  - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
  - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
  - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**戻り値:**

戻り値は0または-1です。0は設定成功を示し、-1は設定失敗を示します。

**例:**

```
-- アドレス2048から16ビットの符号なし整数を1つ書き込みます。
local data = {12}
SetHoldRegs(id, 2048, 1, data)
```

```
-- アドレス2048から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")
```

```
-- アドレス2048から64ビットの倍精度浮動小数点数を2つ書き込みます。
local data = {95.32105, 104.24411}
SetHoldRegs(id, 2048, 8, data, "F64")
```

# バスレジスタ

## コマンドリスト

バスレジスタコマンドは、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、このコマンドセットを対応していません。

| コマンド | 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]。

**戻り値:**

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]。

**戻り値:**

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]。

**戻り値:**

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]。

**戻り値:**

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]。

**戻り値:**

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]。

**戻り値:**

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を対応します。

**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）を対応します。

**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）を対応します。

  浮動小数点数の格納メカニズム (IEEE754) による制限により、単精度浮動小数点数は、(小数点の位置に関係なく) 約 6 ～ 7 の有効数字を格納できます。有効桁数が 6 桁を超える値を単精度浮動小数点数として保存すると、有効桁数が増えるほど、ずれが大きくなる可能性があります。

**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

## コマンドリスト

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。

| コマンド | 機能 |
| --- | --- |
| Print | デバッグ情報をコンソールに出力します。 |
| Log | カスタムログを出力します。 |
| Wait | 指定時間待機するか、指定条件が満たされてから次のコマンドを実行します。 |
| Pause | スクリプトの実行を一時停止します。 |
| Halt | スクリプトの実行を停止します。 |
| ResetElapsedTime | タイマー計測を開始します。 |
| ElapsedTime | タイマー計測を終了します。 |
| Systime | 現在のシステム時間を取得します。 |
| SetGlobalVariable | グローバル変数を設定します。 |
| Popup | スクリプト実行時にポップアップを設定します。 |

## Print

**プロトタイプ:**

```
Print(value)
```

**説明:**

デバッグ情報をコンソールに出力します（コマンド名は `print` と書くこともできます）。

> **説明:**
> 変数の出力フォーマットは、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、出力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：出力待ちのデータ。

**例：**

```
-- コンソールに文字列Successを出力します。
Print('Success')
```

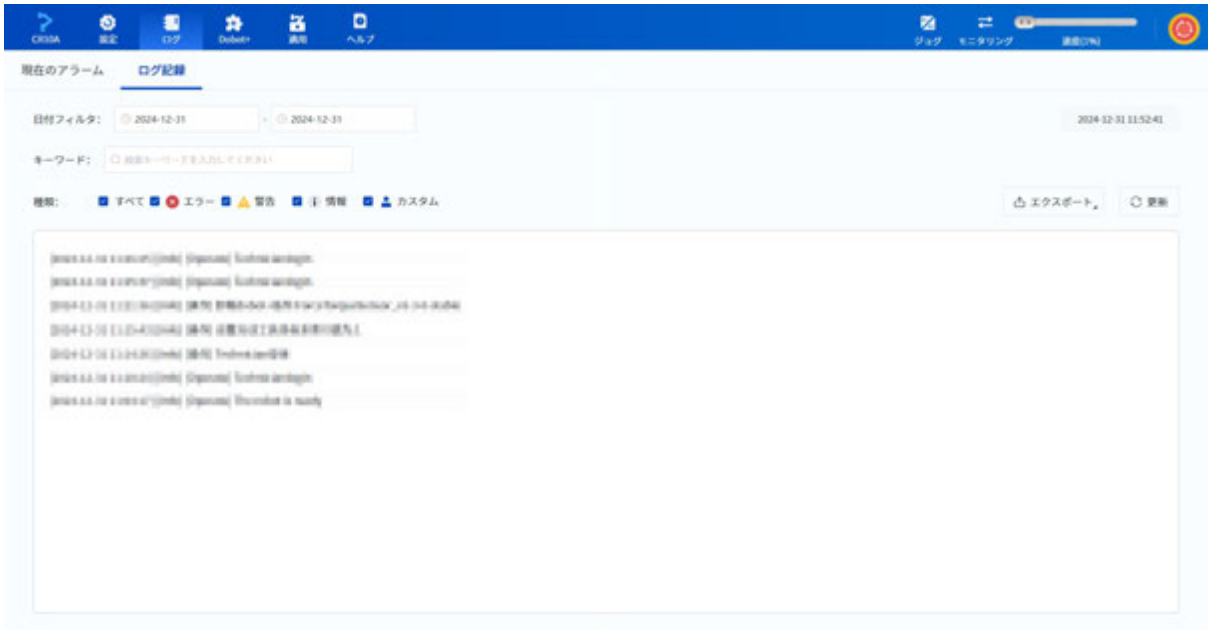
# Log

**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:**

ロボットが前のコマンドを完了すると、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。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コマンドが無効になります。

**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であれは次のコマンドの実行に進みます。

**オプションパラメータ:**

timeout_ms：タイムアウト時間。単位：ms。

- 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、システムは次のコマンドの実行に進み、「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。

**戻り値:**

- 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")
```

```
-- 1秒以内のDI1の状態に基づいて異なるビジネスロジックを実行します。
flag=Wait("DI(1) == ON", 1000)
if(flag==true)
then
    -- DI1の状態がONの場合
else
    -- DI1の状態がOFFで、1秒以上待機した場合
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)
```

# Halt

**プロトタイプ:**

```
Halt()
```

**説明:**

スクリプトの実行を停止します。

**例:**

```
-- 変数countが100の場合、スクリプトの実行を停止します。
if(count==100)
then
    Halt()
end
```

# ResetElapsedTime

**プロトタイプ:**

```
ResetElapsedTime()
```

**説明:**

このコマンドの前のすべてのコマンド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、コンソールに出力します。
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。

**戻り値:**

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。

現地時間を取得するには、現地のタイムゾーンに応じて取得したグリニッジ標準時変換を使用してください。

**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)
```

## Popup

**プロトタイプ:**

```
Popup(message, title, type, logControl)
```

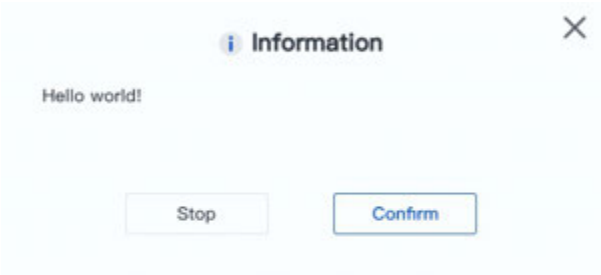
**説明:**

スクリプトの実行中に、ユーザーがカスタマイズ可能な情報を表示するウィンドウをポップアップします。DobotStudio Proでは同時に1つのポップアップのみが表示され、内容は最新のポップアップ情報が反映されます。

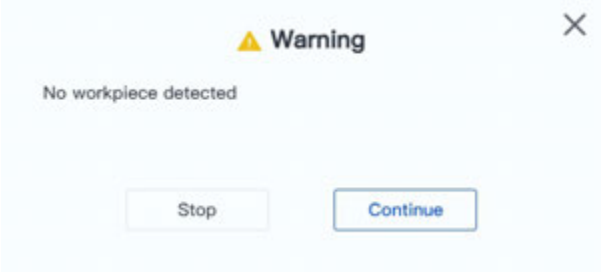
もし、DobotStudio Proを起動せずに（例: IOまたはModbusを通じてプロジェクトを開始した場合）、ポップアップは表示されません。ただし、警告やエラータイプのポップアップコマンドは引き続きプロジェクトを一時停止させます。

**必須パラメータ:**

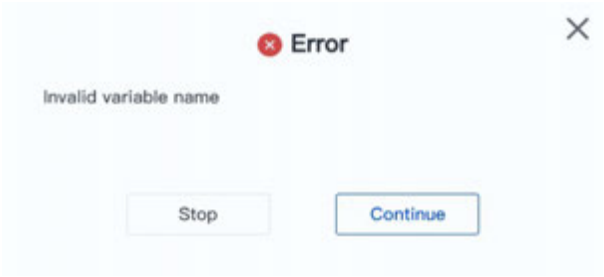
- **message**：表示する情報の内容。文字列を含むさまざまなlua変数をサポートします。 文字列の場合、長さは128文字を超えてはなりません。他の変数タイプの場合、文字列に変換され、変換後の文字サイズは512バイトを超えてはなりません。
- **title**：表示するポップアップのタイトル。文字列のみをサポートし、長さは128文字を超えてはなりません。
- **type**：メッセージタイプ。
  - 0：情報。このタイプのポップアップはプロジェクトの実行に影響を与えません。**停止**をクリックするとプロジェクトが停止し、**OK**をクリックするとポップアップが閉じます。 ポップアップが表示された後でも、ユーザーはIOやModbusを介してプロジェクトを停止できます。プロジェクトが停止するとポップアップは自動的に消えます。

  - 1：警告。このタイプのポップアップはプロジェクトを一時停止します。**停止**をクリックするとプロジェクトが停止し、**続行**をクリックするとプロジェクトが再開されます。ポップアップが表示された後でも、ユーザーはIOやModbusを介してプロジェクトを停止または続行できます。プロジェクトが停止または続行されると、ポップアップは自動的に消えます。

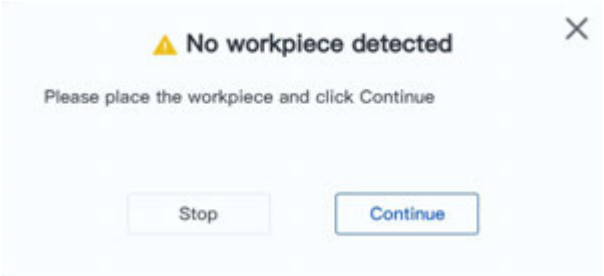
  - 2：エラー。このタイプのポップアップはプロジェクトを一時停止します。**停止**をクリックするとプロジェクトが停止し、**続行**をクリックするとプロジェクトが再開されます。ポップアップが表示された後でも、ユーザーはIOやModbusを介してプロジェクトを停止または続行できます。プロジェクトが停止または続行されると、ポップアップは自動的に消えます。




- **logControl**：ポップアップの内容をログに記録するかどうか。
  - 0：ログに記録しない。
  - 1：対応するタイプのログに記録する（「情報」タイプのポップアップメッセージは「カスタム」タイプのログに対応）。ログの内容は「ポップアップタイトル：ポップアップ内容」となります。

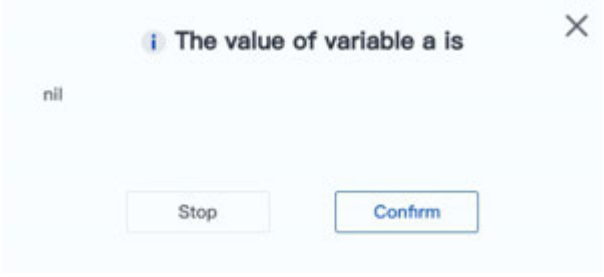
**例1：**

```
-- 警告タイプのポップアップ。ポップアップ内容は文字列で、ログに記録されます。
Popup("ワークを配置してから続行をクリックしてください", "ワークが検出されません", 1, 1)
```




**例2：**

```
-- 情報タイプのポップアップ。ポップアップ内容は変数で、ログには記録されません。
Popup(a, "変数aの値は", 0, 0)
```



# トレイ

## コマンドリスト

トレイは、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、トレイ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。トレイコマンドを使用すると、少数の点を教示するだけで完全なトレイ点配列を作成でき、作成したトレイ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。

| コマンド | 機能 |
| --- | --- |
| CreateTray | トレイの作成 |
| GetTrayPoint | トレイ点を取得する |

## CreateTray

**プロトタイプ:**

```
CreateTray(Trayname, {Count}, {P1,P2})  -- 1次元トレイ
CreateTray(Trayname, {row,col}, {P1,P2,P3,P4})  -- 2次元トレイ
CreateTray(Trayname, {row,col,layer}, {P1,P2,P3,P4,P5,P6,P7,P8})  -- 3次元トレイ
```

**説明:**

トレイを作成し、1次元、2次元、3次元のトレイを作成することができます。

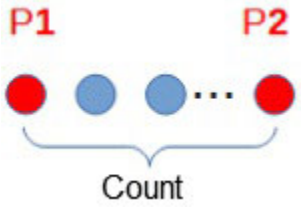
トレイは最大20個まで作成できますが、同名のトレイを作成すると元のトレイが上書きされ、トレイ数は増加しません。

**必須パラメータ:**

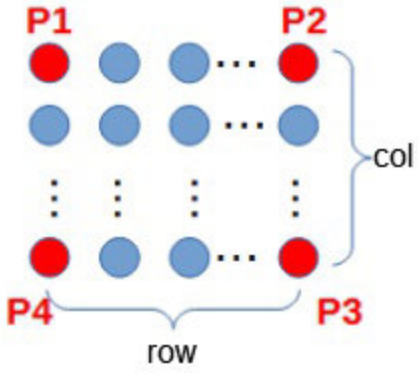
- Trayname：トレイ名。純粋な数字やスペースを使用できない最大32バイトの文字列。

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トレイの次元によって異なります。以下にそれぞれ紹介します。

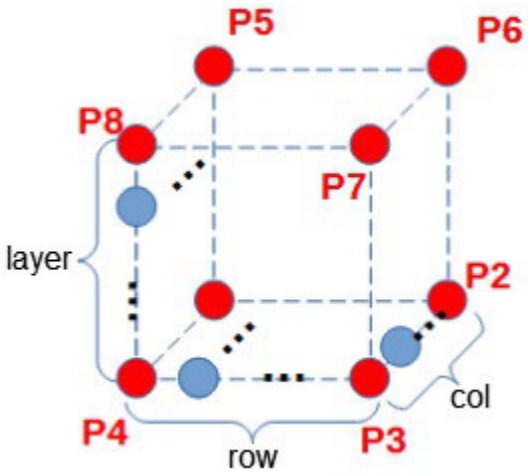
- 1次元トレイの作成：1次元トレイは1直線に沿って等間隔に配置された点のセットです。
    - {Count}：Countはポイントの数を表し、値の範囲は[2,50]で、整数以外の数値を入力すると、自動的に切り捨てられます。
    - {P1、P2}：P1とP2はそれぞれ1ビットトレイの2つのエンドポイントであり、教示点と姿勢変数をサポートします。




- 2次元トレイの作成：2次元トレイ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次元トレイのCountと同じです。
  - {P1、P2、P3、P4}：P1、P2、P3、P4はそれぞれ2次元トレイの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次元トレイの作成：3次元トレイは、空間内に立体的に分散された点の集合であり、複数の2次元トレイ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（P1～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次元トレイの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次元トレイを作成します。
CreateTray("t1", {5}, {P1,P2})
-- t2という名前の4x5の2Dトレイを作成します。
CreateTray("t2", {4,5}, {P1,P2,P3,P4})
-- t3という名前の4x5x6の3次元トレイを作成します。
CreateTray("t2", {4,5,6}, {P1,P2,P3,P4,P5,P6,P7,P8})
```

## GetTrayPoint

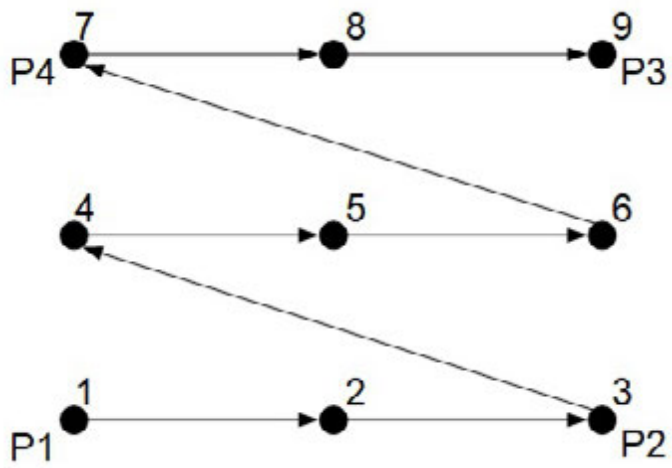
**プロトタイプ:**

```
GetTrayPoint(Trayname, index)
```

**説明:**

指定トレイの指定された番号の点位置を取得します。点の番号はトレイの作成時に渡された点の順番に関わります。

- 1次元トレイ：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次元トレイ：次の図は、3x3トレイを例として、教示点と点の番号の関係を示しています。




- 3次元トレイ：2次元トレイを参照すると、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トレイ名。
- Index：取得する点位置の番号。

**戻り値:**

番号に対応する点位置の座標。トレイ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。トレイ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という名前のトレイの番号が3のて点位置を取得します。
GetTrayPoint("t1",3)
```

# セフティスキン

## コマンドリスト

セフティスキンは、セフティスキンに関連する機能の設定に使用されます。

| コマンド | 機能 |
| --- | --- |
| EnableSafeSkin | セフティスキンスイッチ |
| SetSafeSkin | セフティスキン各部位の感度を設定します |

## EnableSafeSkin

**プロトタイプ:**

```
EnableSafeSkin(ON|OFF)
```

**説明:**

セフティスキンスイッチ。

**必須パラメータ:**

ON|OFF：ONはセフティスキンをオンにすることを表し、OFFはセフティスキンをオフにすることを表します。

**戻る:**

- 0：セフティスキンは検出されませんでした
- 1：セフティスキンは検出されました

**例:**

```
-- セフティスキンをオンにします。
EnableSafeSkin(ON)
```

## SetSafeSkin

**プロトタイプ:**

```
SetSafeSkin(part,status)
```

**説明:**

セフティスキン各部位の感度を設定します。

**必須パラメータ：**

- part：設定対象の部位。値範囲：3~6
  - 3：前腕部
  - 4：J4関節
  - 5：J5関節
  - 6：J:6関節
- status：設定対象の感度。値範囲：0~3
  - 0：オフ
  - 1：低感度（接近距離≤5cm）
  - 2：中程度感度（接近距離≤10cm）
  - 3：高感度（接近距離≤15cm）

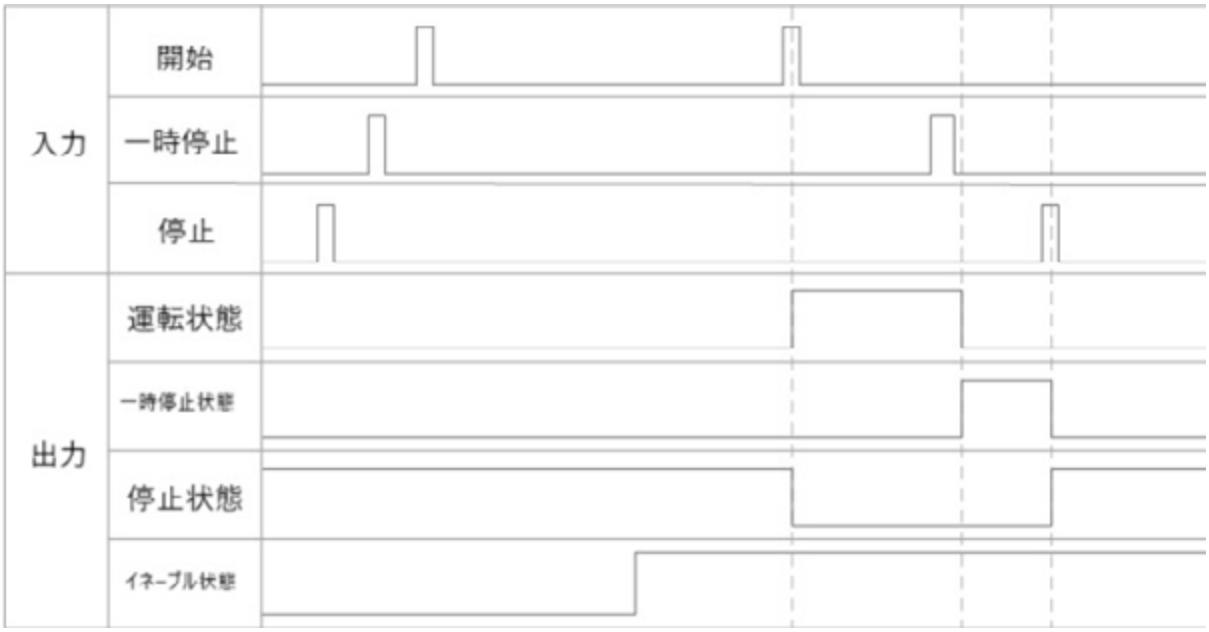
**例：**

```
-- 前腕部のセフティスキン感度をオフに設定します。
SetSafeSkin(3,0)
```

# 付録D リモコン信号タイムチャート

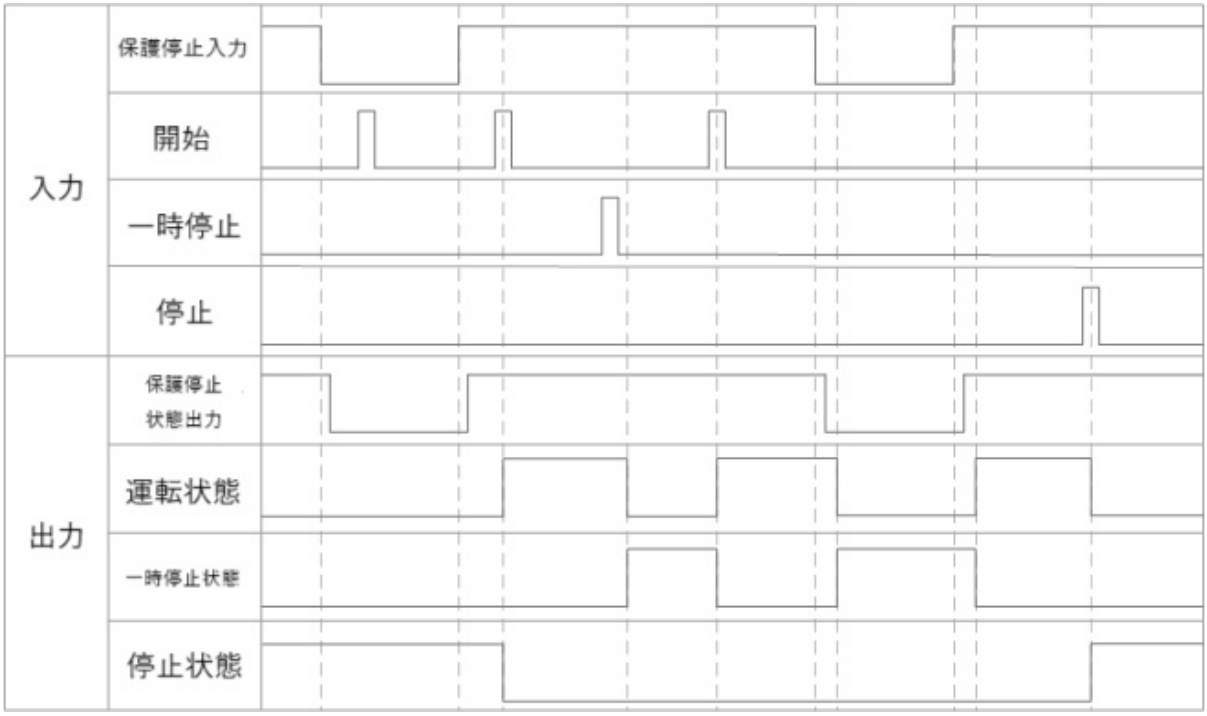
リモート信号を使用しロボットを制御する場合、一般的なタイムチャートは下記の通りです。

ロボットがディセーブル状態の場合、プロジェクト制御関連の信号は有効になりません。




保護停止入力が設定され、保護停止リセット入力 (安全 IO) が設定されていない場合、一般的なタイムチャートは下記の通りです。

保護停止状態になった場合、プロジェクト制御関連の信号は有効になりません。ロボットがプロジェクトを実行する際に、保護停止状態になると、プロジェクトが一時停止になりますが、保護停止状態がリセットされた後、プロジェクトの実行を継続します。


入力
保護停止入力
開始
一時停止
停止
出力
保護停止
状態出力
運転状態
一時停止状態
停止状態

# 付録E Pythonプログラミング関数の説明


- E.1 基本文法
- E.2 共通説明
- E.3 モーションコマンド
- E.4 相対運動コマンド
- E.5 モーションパラメータ
- E.6 IO
- E.7 末端ツール
- E.8 TCP&UDP
- E.9 Modbus
- E.10 プログラム制御

# 基本文法

> **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されます。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**モニタリング > グローバル変数**画面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。


下記の例は、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対応しており、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）を対応します。

```
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。

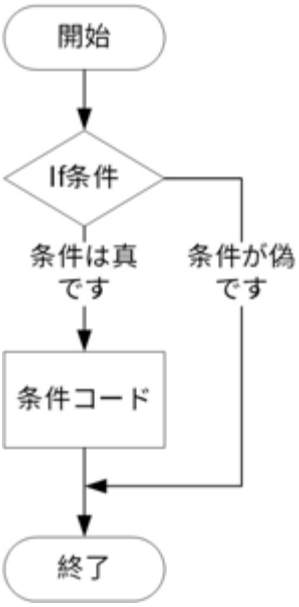
```
dict1 = {"user" : 1, "tool" : 0, "a" : 20, "v" : 50, "cp" : 100}
print(dict1["user"])      # 1
dict1["tool"] = 1         # 辞書内のtoolの値を1に変更します
dict1["r"] = 5            # キーが辞書に存在しない場合は、新しいキーと値のペアが辞書に追加されます。
```

```
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は必須ではありません。

```
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

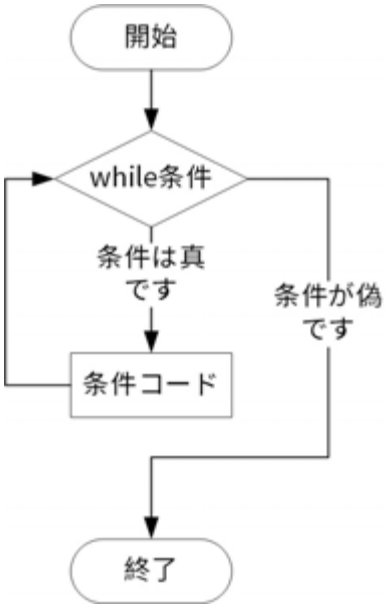
ifステートメントは入れ子にすることができます。典型的な例を次に示します。

```
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
```

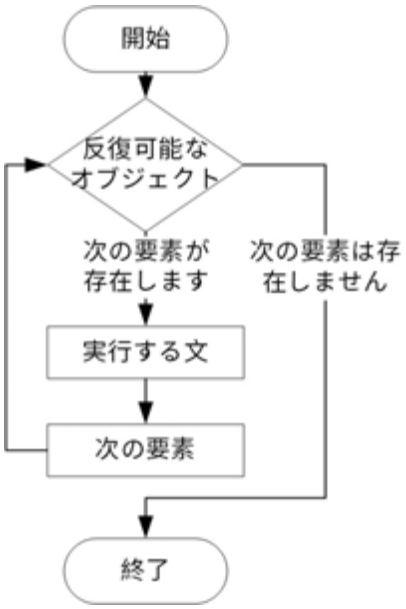
- 「condition」がTrueの場合、「statement_block」ステートメントブロックは実行され、その後「condition」が再判定されます
- 「condition」がFalseの場合、「statement_block」ステートメントブロックはスキップされ、後続のステートメントが直接実行されます

例:

```
a=10
while a < 20
    print("a 的值为:", a)  # 执行10次，输出值为10到19
    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
```

# 共通説明

## 運動方式

ロボットの運動方法は以下に分類されます。

**関節運動**

ロボットは現在点の関節角度と目標点の関節角度の差に基づいて、各関節が同時に動作を完了するように計画します。ジョイントモーションはTCP（Tool Center Point）の動作軌道を制約しません。一般的には動作軌道は非直線となります。


関節運転は特異点によって制限されません （特異点の位置の詳細については、ロボットの対応するハードウェア マニュアルをご参照ください）。そのため、動作軌道の条件がない場合、または目標点が特異点に近い場合は、ジョイントモーションを使用することを推奨します。

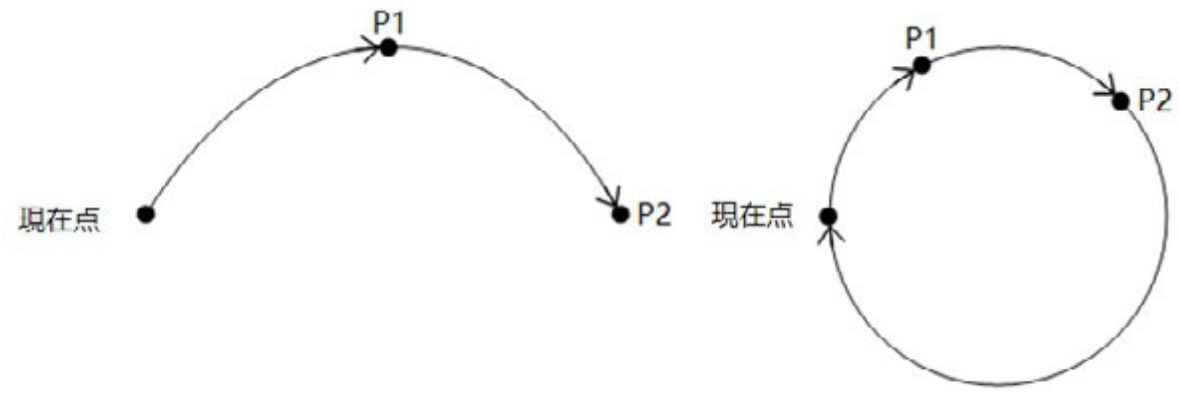
**直線運動**

ロボットは現在点の姿勢と目標点の姿勢に基づいて、TCPの動作軌跡が運動中に等速で変化するように計画します。動作軌道は直線となります。


動作軌跡が特異点を通過する場合、ロボットに直線運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

**円弧運動**

ロボットは、現在点、P1、P2、3つの一直線ではない点によって、円弧または全円を確定します。運動中、ロボットの先端姿勢は、現在点とP2点の姿勢を補間して計算されます。P1点の姿勢は計算に含まれません（つまり、運動中にP1点に到達したときのロボットの姿勢はティーチング時の姿勢と異なる可能性があります）。




動作軌跡が特異点を通過する場合、ロボットに円弧運動コマンドを実行するとエラーが発生します。目標点を再計画するか、特異点の付近に関節動作を使用することを推奨します。

## 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位置パラメータは3種類の表現方法を対応します。

- 関節変数：各ロボットの各関節の角度(j1-j6)を使用して目標位置を表示します。

  ジョイント変数が直線または円弧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。

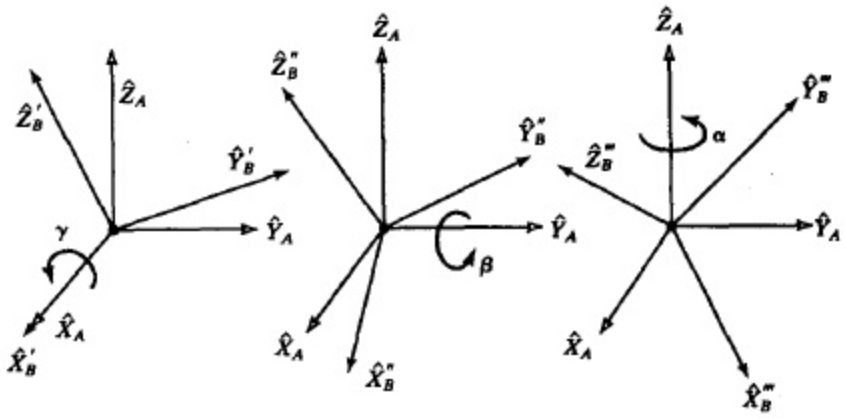
```
{"joint" : [j1, j2, j3, j4, j5, j6]}
```

- 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ツール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。

  ポーズ変数が関節運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（ロボットの現在の関節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、ロボット先端の姿勢を計算します。

- ティーチングポイント：制御ソフトウェアを使用して、ティーチング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：教示時に使用するツール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

運動コマンド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対応しています。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. 運動コマンド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、運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コマンドを参照。コマンド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。
- 関節運動コマンド（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

モーションブロックは、高度な設定にてロボットが当該コマンドを実行する際の加速度（Accel）と速度比率（V）を指定することができます。

```
ロボット実際の運動速度 = 最大速度 x グローバル速度 x コマンド速度比率
ロボット実際の運動加速度 = 最大加速度 x コマンド速度比率
```

最大速度/加速度は**再現設定**によって制限されます。この設定は、ソフトウェアの運動パラメーターで表示され、変更することができます。

グローバル速度は、DobotStudio Proの速度調整（上図右上隅）またはSpeedFactorコマンドで設定できます。

コマンド速度は、運動コマンド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が使用されます（詳細はVelJ、AccJ、VelL、AccLコマンドを参照、コマンド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に関節運動します
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に関節運動します

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に直線運動します
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に直線運動します
```

**絶対速度**

直線および円弧運動コマンドのオプションパラメータ中のspeedは、ロボットがこの運動コマンド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、グローバルの速度比率から影響を受けないですが、**再現設定**の最大速度の制限を受けます（ロボットがリデュースドモードに入っている場合、減速後の最大速度の制限を受けます）。つまり、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:

```
MovL(P1,{"speed":1000})  # 絶対速度1000でP1に直線運動
```

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を目標速度として運動します。

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

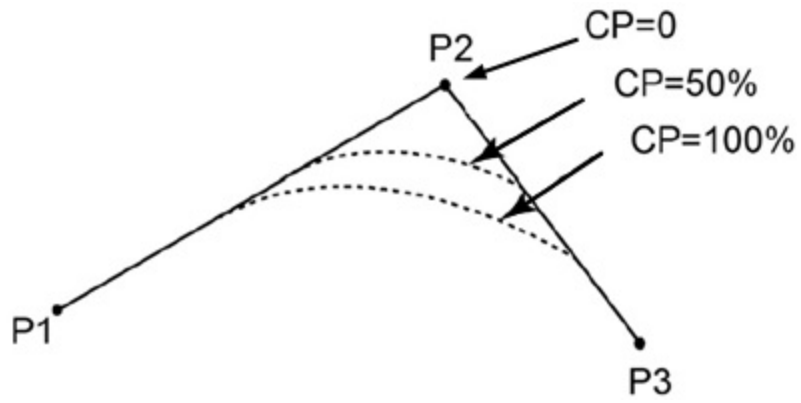
ロボットが複数の点を通過して連続運動する場合、スムーズトランジション方式で中間点を通過し、ロボットを強く回転することを回避することができます。ユーザーが指定する複数の経路点が、異なるツール座標系を基にしている場合、スムーズトランジションを行うことはできません。

オプションパラメータのcpまたはrは、現在の運動コマンドから次の運動コマンドまでのスムーズトランジション比率（cp）またはスムーズトランジション半径（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:**
> 関節運動関連のコマンドはスムーズトランジション半径（r）の設定を対応していません。各コマンドのオプションパラメータをご参照ください。

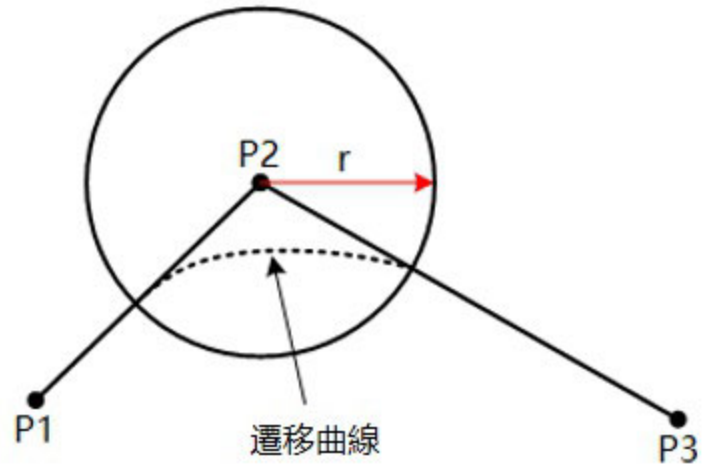
### CP

スムーズトランジション比率を設定する場合は、運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運動曲線は運動速度/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、運動速度/加速度が異なる場合の運動曲線の円弧は異なります。

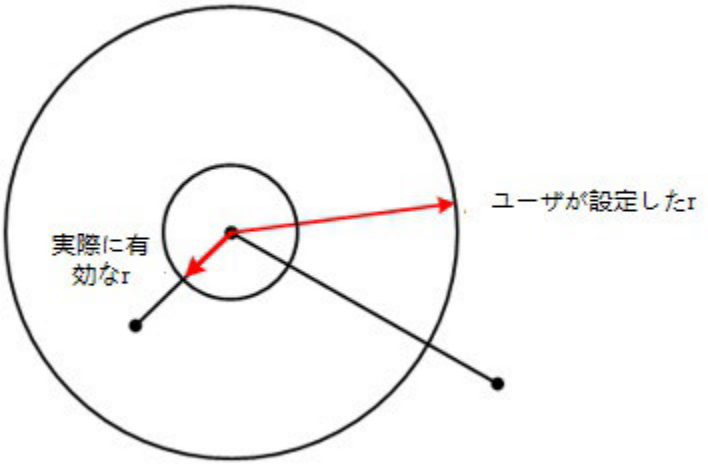



**R**

スムーズトランジションの半径を設定する場合は、運動点を円の中心として、指定半径を基にして運動曲線が計算されます。R運動曲線は運動速度/加速度の影響を受けず、位置と運動半径のみで決定されます。




ユーザーが設定する運動半径が大きすぎる場合は（開始点/終了点と運動点との間の距離を超過）、開始点/終了点と運動点との間の比較的短い距離の半分を運動半径を使用して、運動曲線が計算されます。



オプションパラメータでスムーズトランジション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トランジション比率が使用されます（詳細はCPコマンドを参照、コマンドを呼び出して設定しない場合のデフォルト値は0）。

> **説明:**
> スムーズトランジションにより、ロボットが中間点を通過せずに運動することができます。したがって、スムーズトランジションが設定されている場合、2つの運動コマンドの間の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）コマンドは運動中に実行されます。
> ロボットが正確に中間点に到達したときにコマンドを実行したい場合は、前のコマンドのスムーズトランジションパラメータを0に設定してください。

cpとrの実装方法が異なるため、スムーズトランジションが必要な2つの運動コマンドの間には、運動に影響を与えない他のコマンド(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、コマンドの処理時間が長すぎる場合 (Waitコマンドなど) を除き、ほとんどのコマンドは移行に影響を与えません。

以下の例では、2つの運動コマンドの間にifステートメントを挿入しても、cpモードのスムーズトランジションには影響しません。

```
# 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の場合はスムーズトランジション率50%で次の運動コマンドに移行します。
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コマンドのみがスムーズトランジションに影響しません。他のコマンドを挿入すると、rモードでのスムーズトランジションが無効になります。

```
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つの運動コマンドの間にifステートメントを挿入すると、rモードでのスムーズトランジションの設定が無効になります。

```
# スムーズトランジション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。

# モーションコマンド

## コマンドリスト

モーションコマンドは、ロボットを制御して移動させ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。**

| コマンド | 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 | 関節移動と出力DO |
| MovLIO | 直線移動と出力DO |
| StartPath | 記録した移動軌跡を再生 |
| GetPathStartPose | 軌跡の開始点を取得 |
| PositiveKin | 姿勢に対する関節角度の順解法 |
| InverseKin | 関節角度に対する姿勢の逆解法 |

## 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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]

- v：このコマンドを実行する場合のロボットの移動速度比率。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。値の範囲：[0,100]

詳細は共通説明をご参照ください。

**例：**

```
# ロボット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%、スムーズトランジション制御の比率は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コマンドで呼び出します。動作効果は前のコマンドと同じです。
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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**例:**

```
# ロボット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%、スムーズトランジション制御半径は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コマンドで呼び出します。動作効果は前のコマンドと同じです。
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

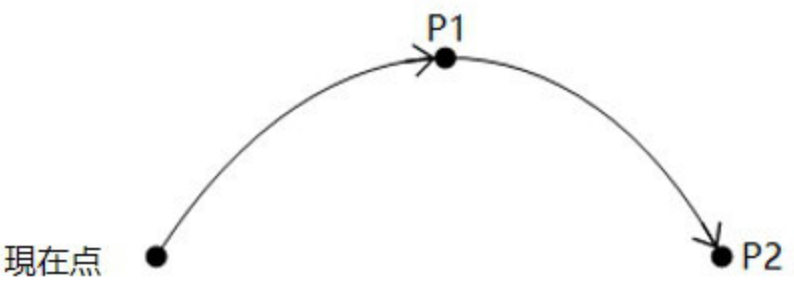
**プロトタイプ:**

```
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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**例:**

```
# ロボットは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

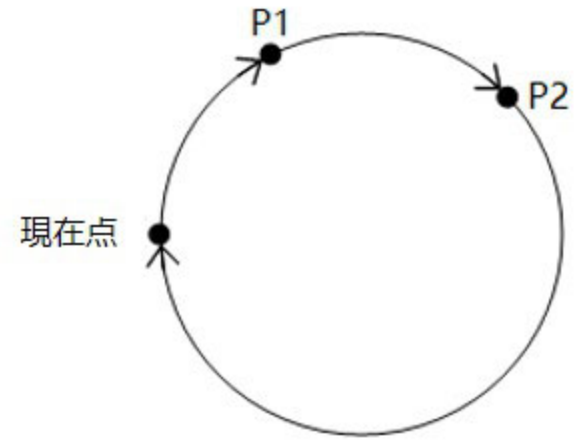
**プロトタイプ:**

```
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、ロボットの移動範囲を超えてはなりません。




移動中のロボットの先端の姿勢は、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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**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:**

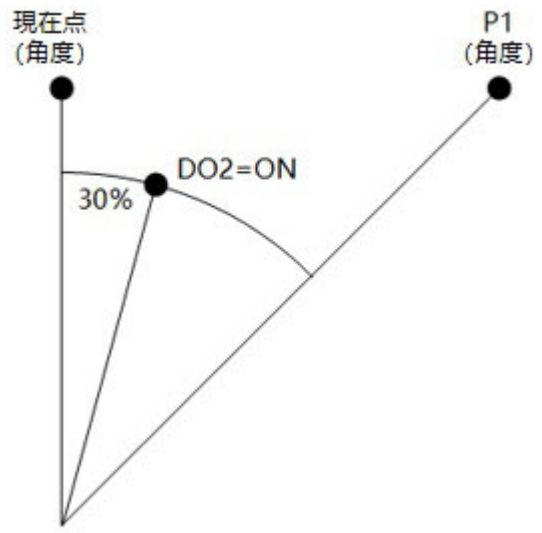
- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。値の範囲：[0,100]

スムーズトランジション制御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。

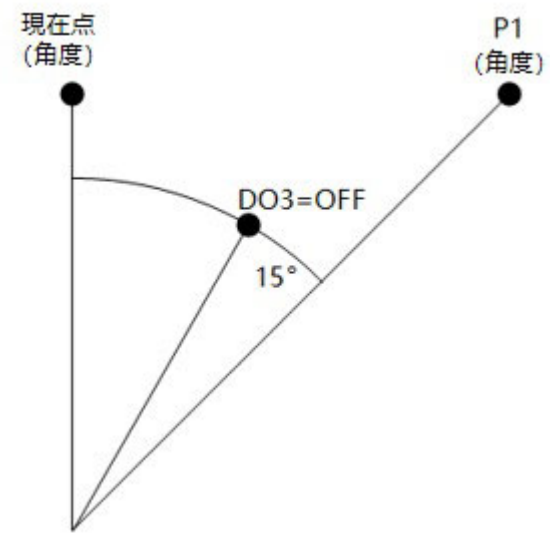
詳細は共通説明をご参照ください。

**例:**

```
# ロボットは、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。

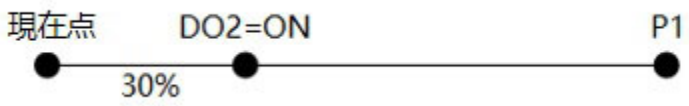
- tool：目標点のツール座標系。
- a：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この指令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

スムーズトランジション制御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。

詳細は共通説明をご参照ください。

**例:**

```
# ロボットは、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。このコマンドを呼び出す前に、ユーザー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]、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、元の軌跡のスムーズトランジション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：`{"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。

**戻り値**

`{"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：`{"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。

### 戻り値

- 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。

# 相対運動コマンド

## コマンドリスト

相対移動コマンド関数はロボット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。**

| コマンド | 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：偏移する指定点。
- [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。

**戻り値:**

- 指定点がティーチングポイントの場合は、偏移後のティーチングポイントの定数を返します。
- 指定点がジョイント変数または姿勢変数の場合、偏移後の姿勢変数を返します: {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：偏移する指定点。
- [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。

**戻り値:**

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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。値の範囲：[0,100]

詳細は共通説明をご参照ください。

**例:**

```
# ロボット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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：このコマンド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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**例：**

```
# ロボット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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率。値の範囲：(0,100]
- Cp：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]

詳細は共通説明をご参照ください。

**例：**

```
# ロボット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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：このコマンド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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**例:**

```
# ロボット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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率。値の範囲：(0,100]
- cp：スムーズトランジション制御の比率。値の範囲：[0,100]

詳細は共通説明をご参照ください。

**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：偏移する指定点。
- 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°

**戻り値:**

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)
```

# モーションパラメータ

## コマンドリスト

モーションパラメータ関数は、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されます。**使用する前に共通説明を読んでください。**

| コマンド | 機能 |
| --- | --- |
| CP | スムーズトランジション制御の比率の設定 |
| 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 | ロボットのリアルタイム姿勢の取得 |
| GetAngle | ロボット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 | 関節移動コマンドの実行可能性の確認 |
| CheckMovL | 直線または円弧移動コマンドの実行可能性の確認 |
| SetSafeWallEnable | 指定した安全壁のオンまたはオフ |
| SetWorkZoneEnable | 指定の安全エリアのオン・オフ |
| SetCollisionLevel | 衝突レベルの設定 |
| SetBackDistance | 衝突戻り距離の設定 |

## CP

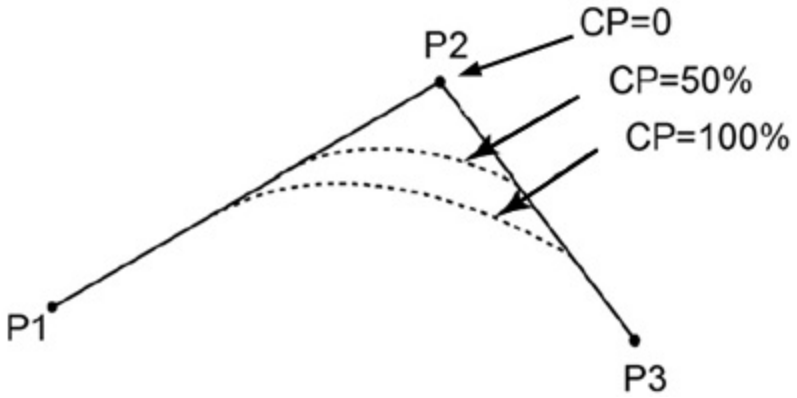
**プロトタイプ:**

```
CP(R)
```

**説明:**

スムーズトランジション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スムーズトランジション制御の比率は、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：スムーズトランジション制御の比率。値の範囲：[0, 100]

**例:**

```
# ロボットは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:**

ロボットの全体的な移動速度を設定します。詳細はモーションコマンド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:**

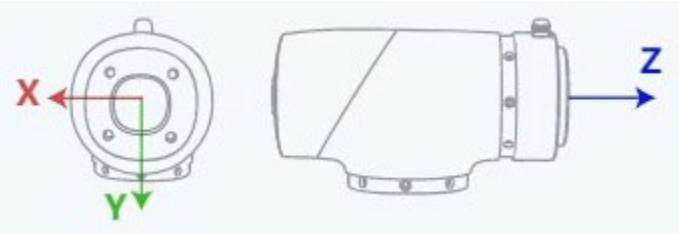
```
SetPayload(payload, [x, y, z])
SetPayload(name)
```

**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。ユーザーがその他コマンドを呼び出す場合、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,type)
```

**説明:**

指定されたユーザー座標系を変更します。

**必須パラメータ:**

- **index**: ユーザー座標系のインデックス。このインデックスは、制御ソフトウェアであらかじめ追加された座標系である必要があります。指定された座標系インデックスが存在しない場合、プログラムの実行が停止し、エラーが発生します。
- **table**: 変更後のユーザー座標系を[x, y, z, rx, ry, rz]の形式で指定します。rx、ry、rzの回転値が含まれる場合、CalcUserコマンドを使用して値を取得することを推奨します。

**オプションパラメータ:**

- type

  : グローバル保存の設定。

  - **0**: このコマンドで変更された座標系は、現在のプログラム実行中のみ有効で、プログラム停止後は元の値に戻ります。
  - **1**: このコマンドで変更された座標系はグローバルに保存され、プログラム停止後も変更後の値が保持されます。ただし、`type=1`（グローバル保存）の場合、ユーザーは手動で対応する座標系設定画面に切り替え、再度戻ることで更新後の値を確認する必要があります（手動で画面をリフレッシュする必要があります）。

**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,0)
```

## 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。

**戻り値:**

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。ユーザーがその他コマンドを呼び出す場合、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,type)
```

**説明:**

指定されたツール座標系を変更します。

**必須パラメータ:**

- **index**: ツール座標系のインデックス。このインデックスは、制御ソフトウェアであらかじめ追加された座標系である必要があります。指定された座標系インデックスが存在しない場合、プログラムの実行が停止し、エラーが発生します。
- **table**: 変更後のツール座標系を[x, y, z, rx, ry, rz]の形式で指定します。rx、ry、rzの回転値が含まれる場合、CalcToolコマンドを使用して値を取得することを推奨します。

**オプションパラメータ:**

- type

  : グローバル保存の設定。

  - **0**: このコマンドで変更された座標系は、現在のプログラム実行中のみ有効で、プログラム停止後は元の値に戻ります。
  - **1**: このコマンドで変更された座標系はグローバルに保存され、プログラム停止後も変更後の値が保持されます。ただし、`type=1`（グローバル保存）の場合、ユーザーは手動で対応する座標系設定画面に切り替え、再度戻ることで更新後の値を確認する必要があります（手動で画面をリフレッシュする必要があります）。

**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,0)
```

# 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。

**戻り値:**

計算された[x、y、z、rx、ry、rz]形式のツール座標系です。

**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:**

ロボットのリアルタイムの姿勢を取得します。

2つの移動コマンド間でこのコマンドを呼び出す場合、得られるポイントはスムーズトランジション制御設定の影響を受けます。

- スムーズトランジション制御機能（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。
- スムーズトランジション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。

**戻り値:**

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:**

ロボットのリアルタイムの関節角度を取得します。

2つの移動コマンド間でこのコマンドを呼び出す場合、得られる間接角度はスムーズトランジション制御設定の影響を受ける場合があります。

- スムーズトランジション制御機能（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。
- スムーズトランジション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。

**戻り値:**

ロボットの現在の関節角度: `{"joint": [j1、j2、j3、j4、j5、j6]}`

**例:**

```
# ロボットはまず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
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:**

現在の位置から目標点へのジョイントモーションの実行可能性を確認します。システムは移動軌跡全体を計算し、軌跡内に到達不可能なポイントがないかを検査します。

**必須パラメータ:**

- **P**: 目標点。

**オプションパラメータ:**

- **user**: 目標点のユーザー座標系。
- **tool**: 目標点のツール座標系。
- **a**: このコマンドを実行する際のロボットの加速度比率。取値範囲：(0,100]。
- **v**: このコマンドを実行する際のロボットの速度比率。取値範囲：(0,100]。
- **cp**: スムーズな移行の比率。取値範囲：[0,100]。

詳細は一般的な説明を参照してください。

**戻り値:**

検査結果：

- 0: エラーなし
- 16: 終点が肩の特異点に近い
- 17: 終点で逆計算で解なし
- 18: 終点が逆計算で範囲外
- 22: 姿勢切替エラー
- 26: 終点が手首の特異点に近い
- 27: 終点が肘の特異点に近い
- 29: 速度パラメータエラー
- 30: 全てのパラメータの逆計算が失敗
- 32: 軌跡に肩の特異点がある
- 33: 軌跡に逆計算が存在しないポイントがある
- 34: 軌跡に逆計算範囲外のポイントがある
- 35: 軌跡に手首の特異点がある
- 36: 軌跡に軸の特異点がある
- 37: 軌跡にジョイントのジャンプポイントがある

**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：このコマンドを実行する場合のロボットの移動加速度比率。値の範囲：(0,100]
- v：このコマンドを実行する場合のロボットの移動速度比率で、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：スムーズトランジション制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズトランジション制御半径で、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

詳細は共通説明をご参照ください。

**戻り値:**

検査結果。

- 0: エラーなし
- 16: 終点が肩の特異点に近い
- 17: 終点で逆計算で解なし
- 18: 終点が逆計算で範囲外
- 22: 姿勢切替エラー
- 26: 終点が手首の特異点に近い
- 27: 終点が肘の特異点に近い
- 29: 速度パラメータエラー
- 30: 全てのパラメータの逆計算が失敗

- 32: 軌跡に肩の特異点がある
- 33: 軌跡に逆計算が存在しないポイントがある
- 34: 軌跡に逆計算範囲外のポイントがある
- 35: 軌跡に手首の特異点がある
- 36: 軌跡に軸の特異点がある
- 37: 軌跡にジョイントのジャンプポイントがある

**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:**

指定された安全壁をオンまたはオフにします。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：安全壁をオンにします。
  - False：安全壁をオフにします。

**例:**

```
# インデックスが1の安全壁をオンにします。
SetSafeWallEnable(1,True)
```

## SetWorkZoneEnable

**プロトタイプ:**

```
SetWorkZoneEnable(index,value)
```

**説明:**

指定した安全エリア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安全エリアの状態は、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：設定する安全エリアのインデックス。事前に制御ソフトウェアで対応する安全エリアを追加する必要があります。値の範囲：[1,6]
- value：
  - True：安全エリアをオンにします。
  - False：安全エリアをオフにします。

**例:**

```
# インデックス1の安全エリアを開きます。
SetWorkZoneEnable(1,True)
```

## SetCollisionLevel

**プロトタイプ:**

```
SetCollisionLevel(level)
```

**説明:**

衝突検知レベルを設定します。CRAモデルの場合、このコマンドは安全制限のTCP力と電力値を設定します。このコマンドで設定された値は、現在のプロジェクト実行中のみ有効であり、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**: 衝突検知レベル。0は衝突検知を無効化、1～5は数字が大きい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

## コマンドリスト

IOコマンドは、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。

| コマンド | 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 |

## 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。

**戻り値:**

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。

**戻り値:**

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、コマンド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コマンドが実行されます。

**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。

**戻り値:**

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。

**戻り値:**

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)
```

# 末端ツール

## コマンドリスト

末端ツールコマンドは、ロボットシステムの末端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されます。

| コマンド | 機能 |
| --- | --- |
| ToolDI | 末端デジタル入力ポートの状態を読み取ります |
| ToolDO | 末端デジタル出力ポートの状態を設定します(キューコマンド) |
| GetToolDO | 末端デジタル出力ポートの現在の状態を取得します |
| SetToolPower | 末端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:**

末端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：末端DI端子の番号。

**戻る:**

対応するDI端子の状態（ON/OFF）。

**例:**

```
# 末端DI1が0であるとき、ロボットが直線移動方式で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:**

末端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：末端DO端子の番号。
- ON|OFF：設定するDOポートの状態。

**例:**

```
# 末端DO1をONに設定します。
ToolDO(1,ON)
```

## GetToolDO

**プロトタイプ:**

```
GetToolDO(index)
```

**説明:**

末端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：末端DO端子の番号。

**戻る**

対応する末端DO端子の状態（ON/OFF）。

**例:**

```
# 末端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:**

末端ツールの電源供給状態を設定します。通常、末端電源を再起動するために使用されます。例えば、末端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：末端ツールの電源状態。

- 0：電源オフ
- 1：電源オン

**例：**

```
# 末端ツールの電源を入れ直します。
SetToolPower(0)
Wait(5)
SetToolPower(1)
```

# TCP&UDP

## コマンドリスト

TCP&UDP関数は、TCPやUDP通信を行うために使用されます。

| コマンド | 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

**戻り値:**

- 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。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。

**戻り値:**

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。

**戻り値:**

- 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。

**戻り値:**

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。

**戻り値:**

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：サーバーを作成するかどうか。
  - True：サーバーの作成を示します。
  - False：クライアントの作成を示します。
- 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：
  - サーバーを作成するときは、ローカル側と反対側の両方がこのポートを使用することを意味します。すでにシステムが占有しているポートは使用しないでください。詳細については、TCPCreate のパラメータの説明を参照してください。
  - クライアントを作成する場合、ピアのポートを示します。このとき、ローカル エンドはデータを送信するときにランダムなポートを使用します。。

**戻り値:**

- 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1：**

```
# 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(True, ip, port)
```

**例2：**

```
# UDP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。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。

**戻り値：**

- 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。

**戻り値:**

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

## コマンドリスト

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。

| コマンド | 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 | 単一書き込み | 連続書き込み |
| --- | --- | --- | --- |
| 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。詳細は付録A Modbusデー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。

**オプションパラメータ：**

- slave_id：スレーブID。
- isRTU：ブール値。
    - isRTUがfalseの場合、コントローラーのネットワークポートによるModbus TCP通信を確立します。
    - isRTUがtrueの場合、ロボット末端のRS485によるModbus RTU通信を確立します。ポート60000のみが使用可能です。

> ⚠ **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。

**戻り値：**

- 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コマンド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:**

コントローラーのRS485インターフェースによるModbusマスターを作成し、スレーブとの接続を確立します。同時に最大で15のデバイスとの接続を対応します。

**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。

**オプションパラメータ:**

- parity：パリティビットかどうか。
  - 「O」：奇数パリティ
  - 「E」：偶数パリティ
  - 「N」：パリティビットなし
  - 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。

**戻り値:**

- 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、後でその他Modbusコマンドを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。

**戻り値:**

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スレーブのコイルレジスタアドレスの値を読み取ります。対応するModbus機能コードは02です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: コイルレジスタの開始アドレス。
- **count**: 連続して読み取るコイルレジスタの数。取値範囲は [1, 2000] です（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**戻り値:**

コイルレジスタアドレスの値がtable形式で返されます。tableの最初の値はコイルレジスタの開始アドレスに対応する値です。

**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:**

指定されたデータ型に従い、Modbusスレーブの入力レジスタアドレスの値を読み取ります。対応するModbus機能コードは04です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 入力レジスタの開始アドレス。
- **count**: 連続して読み取る入力レジスタの数。取値範囲は [1, 125]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**オプションパラメータ:**

- type: データ型。
    - 空の場合、デフォルトでU16（16ビット符号なし整数）。
    - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
    - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
    - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
    - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**戻り値:**

入力レジスタアドレスの値がtable形式で返されます。tableの最初の値は入力レジスタの開始アドレスに対応する値です。

**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スレーブのコイルレジスタアドレスの値を読み取ります。対応するModbus機能コードは01です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: コイルレジスタの開始アドレス。
- **count**: 連続して読み取るコイルレジスタの数。取値範囲は [1, 2000]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**戻り値:**

コイルレジスタアドレスの値がtable形式で返されます。tableの最初の値はコイルレジスタの開始アドレスに対応する値です。

**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:**

指定された値をコイルレジスタの指定アドレスに書き込みます。対応するModbus機能コードは05（単一書き込み）および0F（複数書き込み）です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: コイルレジスタの開始アドレス。
- **count**: 連続してコイルレジスタに書き込む数。取値範囲は [1, 1968]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。
- **table**: コイルレジスタに書き込む値が格納されたtable。tableの最初の値はコイルレジスタの開始アドレスに対応します。

**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:**

指定されたデータ型に従い、Modbusスレーブの保持レジスタアドレスの値を読み取ります。対応するModbus機能コードは03です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 保持レジスタの開始アドレス。
- **count**: 連続して読み取る保持レジスタの数。取値範囲は [1, 125]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。

**オプションパラメータ:**

- type: データ型。
  - 空の場合、デフォルトでU16（16ビット符号なし整数）。
  - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
  - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
  - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
  - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**戻り値:**

保持レジスタアドレスの値がtable形式で返されます。tableの最初の値は保持レジスタの開始アドレスに対応する値です。

**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:**

指定されたデータ型に従い、指定した値を保持レジスタの指定アドレスに書き込みます。対応するModbus機能コードは06（単一書き込み）および10（複数書き込み）です。

**必須パラメータ:**

- **id**: 作成済みのマスタインデックス。
- **addr**: 保持レジスタの開始アドレス。
- **count**: 連続して保持レジスタに書き込む数。取値範囲は [1, 123]（ModbusTCPプロトコルの制限に基づく。実際の取値範囲はスレーブのレジスタ数やプロトコル仕様に従って決定してください）。
- **table**: 保持レジスタに書き込む値がtable形式で返されます。tableの最初の値は保持レジスタの開始アドレスに対応します。

**オプションパラメータ:**

- type: データ型。
    - 空の場合、デフォルトでU16（16ビット符号なし整数）。
    - **U16**: 16ビット符号なし整数（2バイト、1レジスタを使用）。
    - **U32**: 32ビット符号なし整数（4バイト、2レジスタを使用）。
    - **F32**: 32ビット単精度浮動小数点数（4バイト、2レジスタを使用）。
    - **F64**: 64ビット倍精度浮動小数点数（8バイト、4レジスタを使用）。

**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

## コマンドリスト

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。

| コマンド | 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（コマンド名は `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:**

ロボットが前のコマンドを完了すると、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であれは次のコマンドの実行に進みます。

**選択可能なパラメータ:**

timeout_ms：タイムアウト時間。単位：ms。

- 判定ロジックが常に false で待機時間がこの時間を超えると、システムは次のコマンドの実行を続けて false を返します。
- 0 以下の場合は、待機なしですぐにタイムアウトになることを意味します。
- このパラメータが設定されていない場合は、タイムアウトせず、判定ロジックが true になるまで待機することを意味します。

**戻り値:**

- 条件が満たされ、操作が続行される場合に true を返します。
- 条件が満たされず、タイムアウトにより操作が続行された場合は、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:**

このコマンドの前のすべてのコマンド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。

**戻り値:**

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。

**戻り値:**

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)
```