DobotStudio Proユーザーマニュアル
(CR & Nova)

Shenzhen Yuejiang Technology CO., Ltd.

目次

前書き

目的

本マニュアルは、6軸ロボットアーム（CRおよびNovaシリーズ）用ロボットアーム制御ソフトウェアDobotStudio Proの機能および操作方法などを説明し、ユーザーがDobotStudio Proをよく理解し、それを使用できるようにするためのものです。

対象読者

本マニュアルは下記の方を対象としています。

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

改定記録

日時	改定記録
2024/02/06	ソフトウェアの対応バージョンは2.8.0です。
2023/06/25	ソフトウェアの対応バージョンは2.7.1です。
2023/03/22	初版

標識の定義

本マニュアル中の下記標識は、それぞれ以下の意味を表しています。

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

 説明	本文への強調または補足である追加情報を示します

1 はじめに

DobotStudio Proのご利用をありがとうございます。DobotStudio Proは、越疆がDobotロボットアームに対して開発したPC端末制御ソフトウェアです。機能はシンプルで使いやすく、実用性が高く、画面は簡潔で分かりやすいなどの特徴があります。ユーザーはDobotロボットアームの使い方をすぐに理解することができます。

本マニュアルは主に、DobotStudio Proを使用して6軸ロボットアーム（CRおよびNovaシリーズ）を制御する方法を説明します。

DobotStudio Proがサポートする**OS**は次のとおりです。

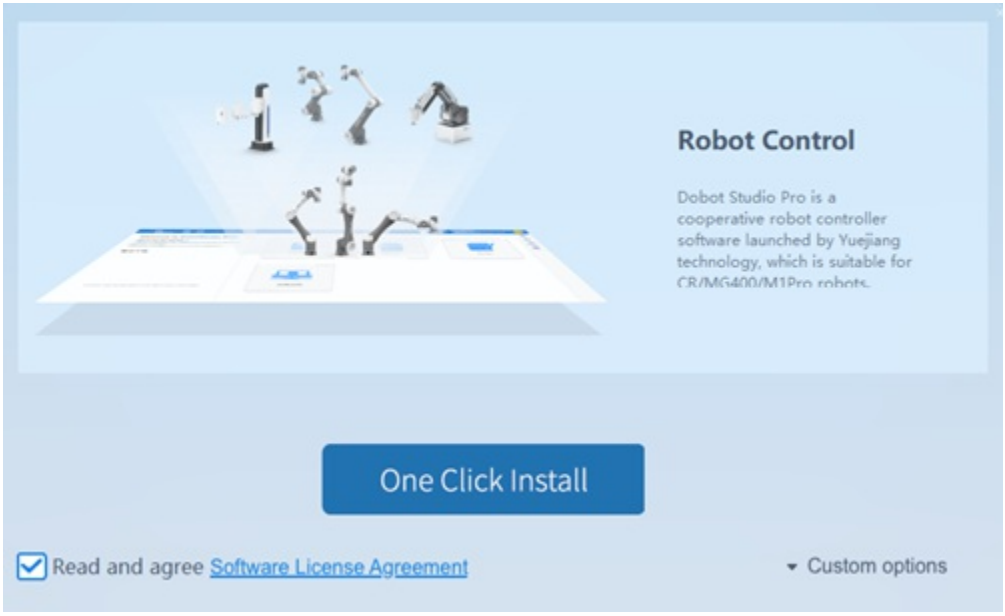
- Win7
- Win10
- Win11

ソフトウェアのインストール

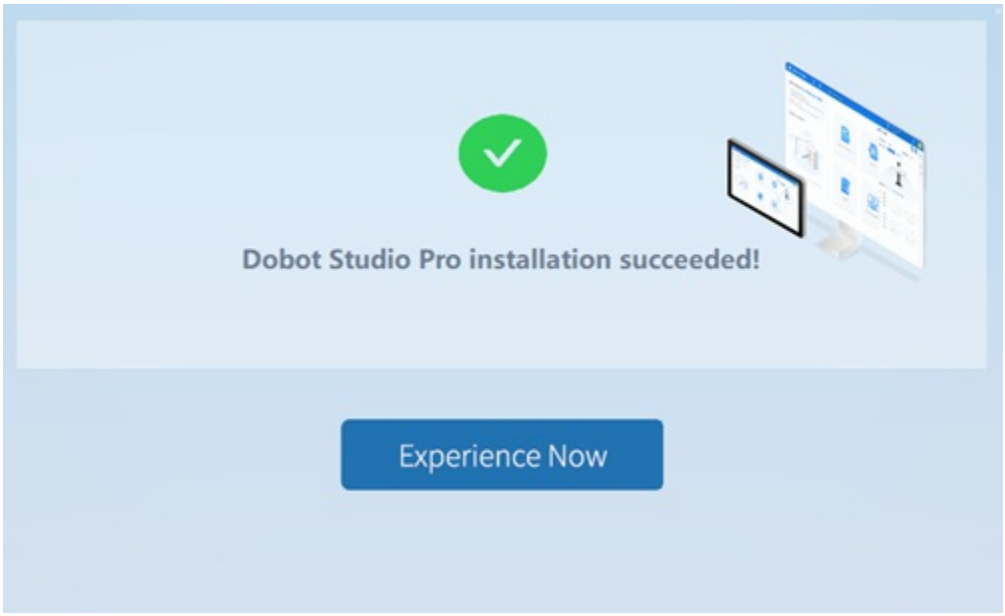
公式ホームページから最新のDobotStudio Proインストールパッケージをダウンロードしてインストールします。インストール手順は次のとおりです。

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

2. インストール画面中で「ワンキーインストール」をクリックするか、カスタムオプションでインストールパスを設定した後、インストールは開始します。



3. ソフトウェアのインストール完了後、プロンプト画面で「今すぐ体験」をクリックすると、直接ソフトウェアを開くことができます。



マニュアルガイド

DobotStudio Proを初めて使用する場合、以下の順序で本マニュアルをご覧いただくことをお勧めします。

1. ロボットの接続：DobotStudio Proをロボットアームに接続します。
2. メイン画面の説明：DobotStudio Proのメイン画面を理解し、DobotStudio Proの主な機能をおおよそ理解します。
3. 設定：実際の必要性によりロボットアームを構成します。ロボットアームが吊り掛け式、壁掛け式の取り付け、あるいは一定角度のある取り付けを採用する場合、イネーブルオフ状態において回転角と傾斜角を設定してください。まず、取り付け設定を参照して構成してくだ

さい。

4. モニタリング：DobotStudio Proが提供するモニタリングなどの機能を理解します。先端プラグインをインストールする場合、先端プラグインを参照して配置してください。
5. プログラミング/プロセス：DobotStudio Proが提供するプログラミングおよびプロセス機能を理解します。自身のプロジェクトを作成してみてください。
6. リモート制御：プロジェクトの開発完了後、リモート制御によりプロジェクトを実行してみてください。

2 ロボットの接続

DobotStudio Proは有線（LAN）と無線（Wi-Fi）の2種類の接続方法に対応しています。

> 📖説明
>
> ロボットコントローラのファームウェアバージョンが3.5.4.0以前である場合、PCでSMB1プロトコルを開くことで、ロボットへ正常に接続できます。詳細は本章の（オプション）SMBプロトコルを開くをご覧ください。

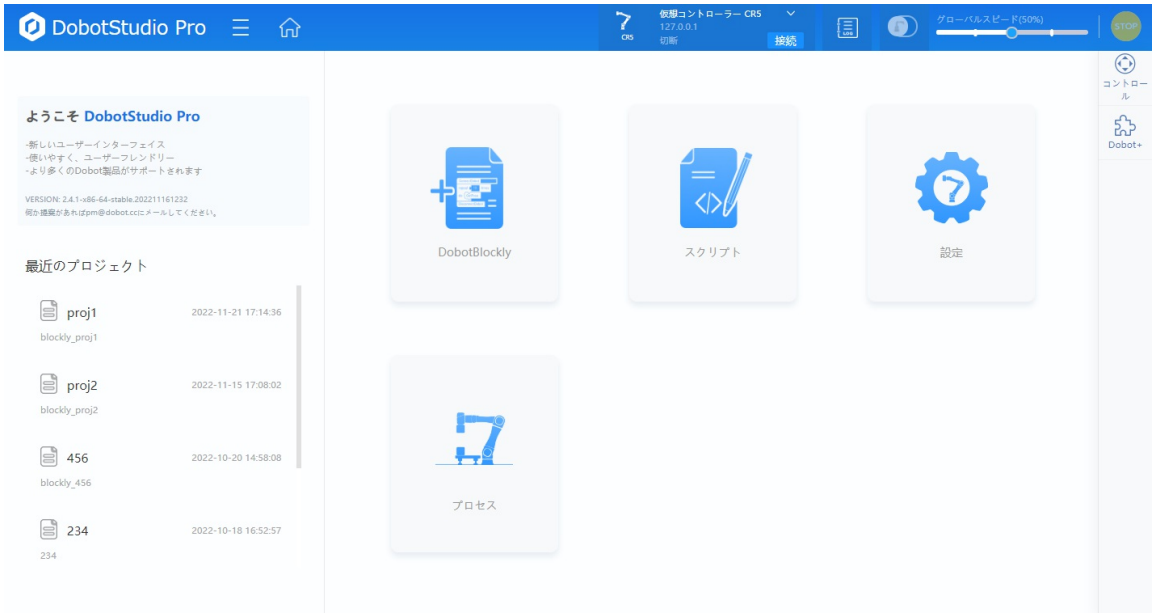
有線接続

ネットワークケーブルの一端をコントロールキャビネットのLANインターフェイスに接続し、他端をPCに接続し、その後、PCのIPアドレスを変更し、それとコントロールキャビネットのIPアドレスを同じネットワークセグメントにしてください。コントロールキャビネットのデフォルトのIPアドレスは192.168.5.1で、通信設定で変更できます。

WindowsのバージョンによってPCのIPアドレスを変更する方法は若干異なるため、本マニュアルはWindows 10を例に紹介します。

1. タスクバーで「ネットワーク」を検索し、「ネットワーク接続を確認」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て「プロパティ」を選択し、その後、ポップアップされたウィンドウから「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）プロパティ」画面で「以下のIPアドレスを使用」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アドレスはコントロールキャビネットと同一ネットワークセグメントのまだ占有されていない任意のIPアドレスに変更でき、そのサブネットマスクおよびデフォルトのゲートウェイはコントロールキャビネットのものと一致します。PCのIPアドレスを192.168.5.100に設定する場合、サブネットマスクは255.255.255.0です。

4. DobotStudio Pro画面の上部で接続するロボットを選択し、その後、「接続」をクリックしてください。



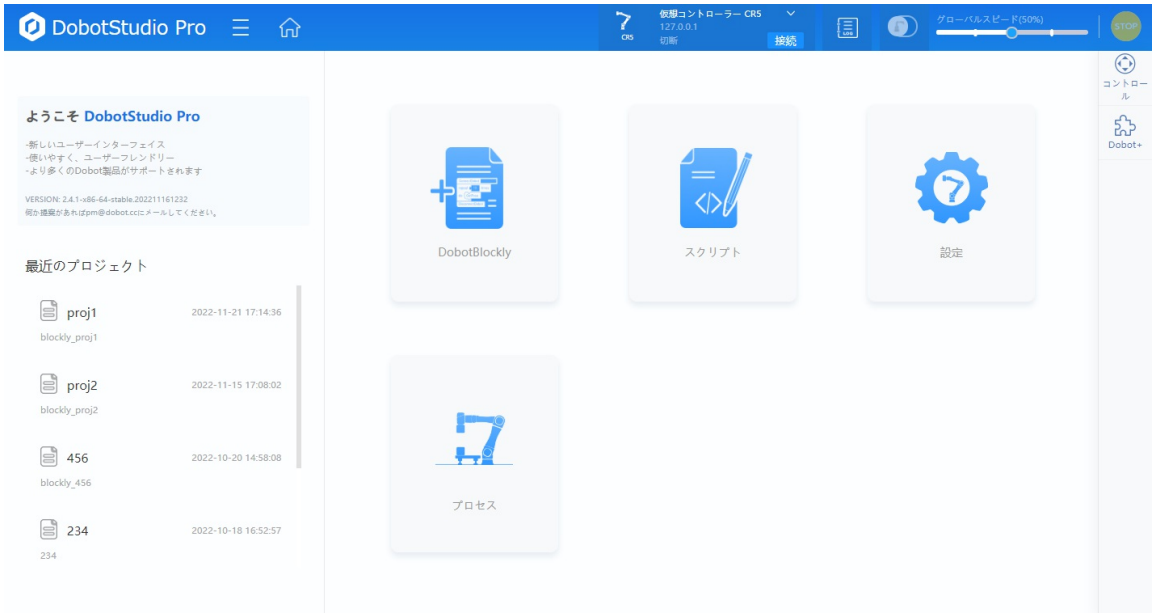
5. 接続が成功すると、DobotStudio Proはプロンプトメッセージをポップアップ表示し、接続状態が「接続済み」に変わります。「接続済み」をクリックすると、ロボットとの接続が解除できます。

無線接続

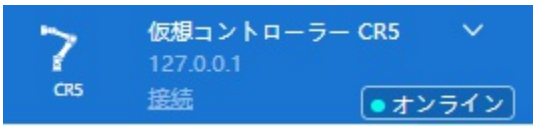
ロボットを接続する前に、コントロールキャビネットにWi-Fiモジュールが既にインストールされていることを確保する必要があります。

1. PC端末でロボットのWi-Fiを検索し接続します。Wi-Fiのデフォルト SSIDは「Dobot{製品モデル}-XXXX-XXXX」で、そのうち、XXXX-XXXXはロボットアームの台座に位置するロボットアーム番号です。初期のWi-Fiのパスワードは「1234567890」です。Wi-FiのSSIDとパスワードは通信設定で変更できます。

2. DobotStudio Pro画面の上部で接続するロボットを選択し、その後、「接続」をクリックしてください。

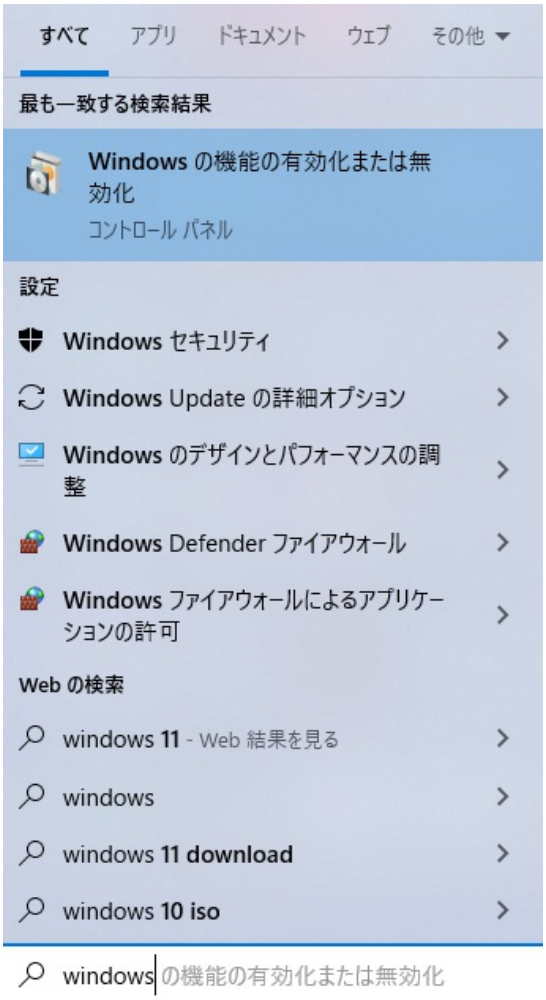


3. 接続が成功すると、DobotStudio Proはプロンプトメッセージをポップアップ表示し、接続状態が「接続済み」に変わります。「接続済み」をクリックすると、ロボットとの接続が解除できます。

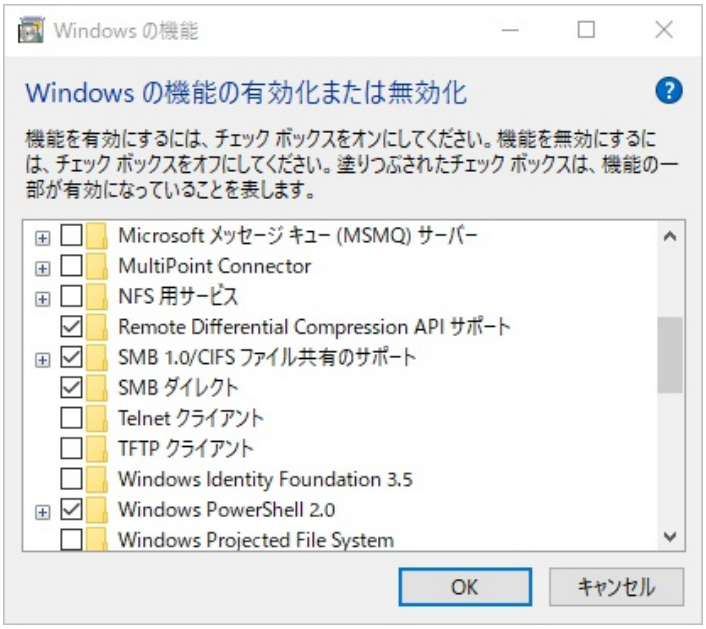


（オプション）SMBプロトコルを開く

1. Windows 10を例に挙げると、タスクバーで「Windows機能」を検索し、「Windows機能を起動または終了」をクリックして開きます。

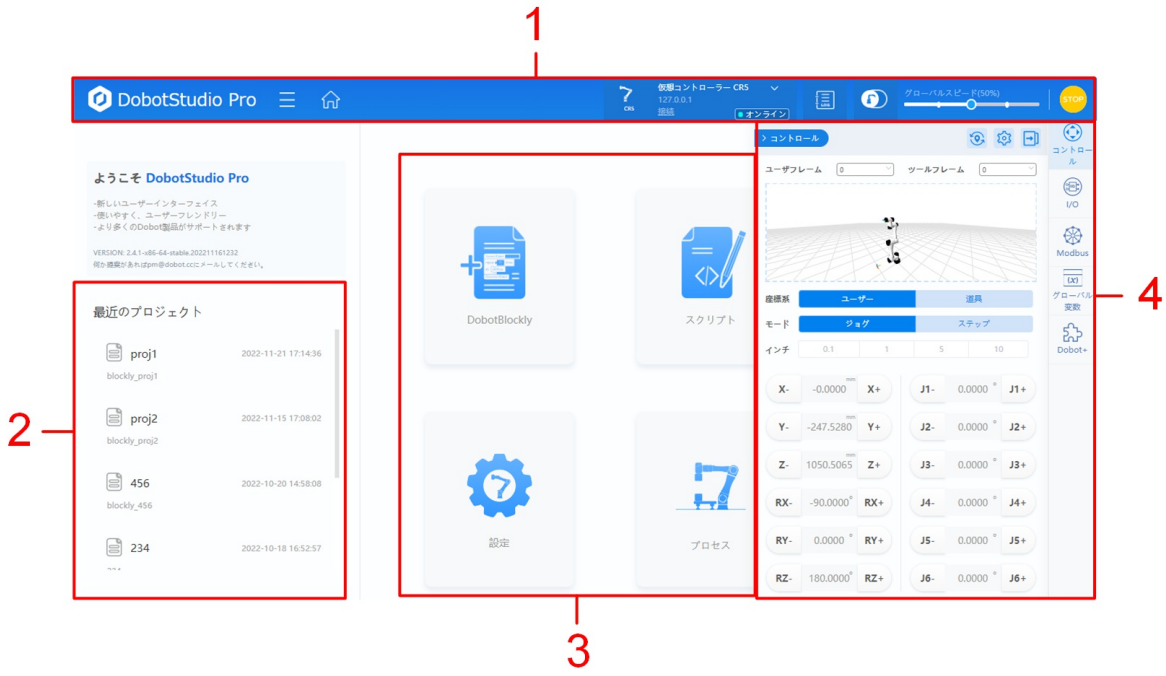


2. 「SMB 1.0/CIFSファイルの共有サポート」をチェックし、「確定」をクリックします。



3 メイン画面の説明

3.1 メイン画面の概要



番号	説明
1	トップツールバー
2	最近開いたプロジェクトを表示します。クリックすると、すばやくプロジェクトを開くことができます。
3	主な機能のアクセスには、グラフィックプログラミング、スクリプトプログラミング、設定、プロセスが含まれています。
4	右のツールバーのボタンをクリックすると、対応するツールパネルが表示、または非表示にされます。上から下にそれぞれ、制御パネル、I/Oモニタリングパネル、Modbusパネル、グローバル変数パネル、Dobot+先端プラグインパネルです。そのうち、制御パネルは接続完了後にデフォルトで表示されます。

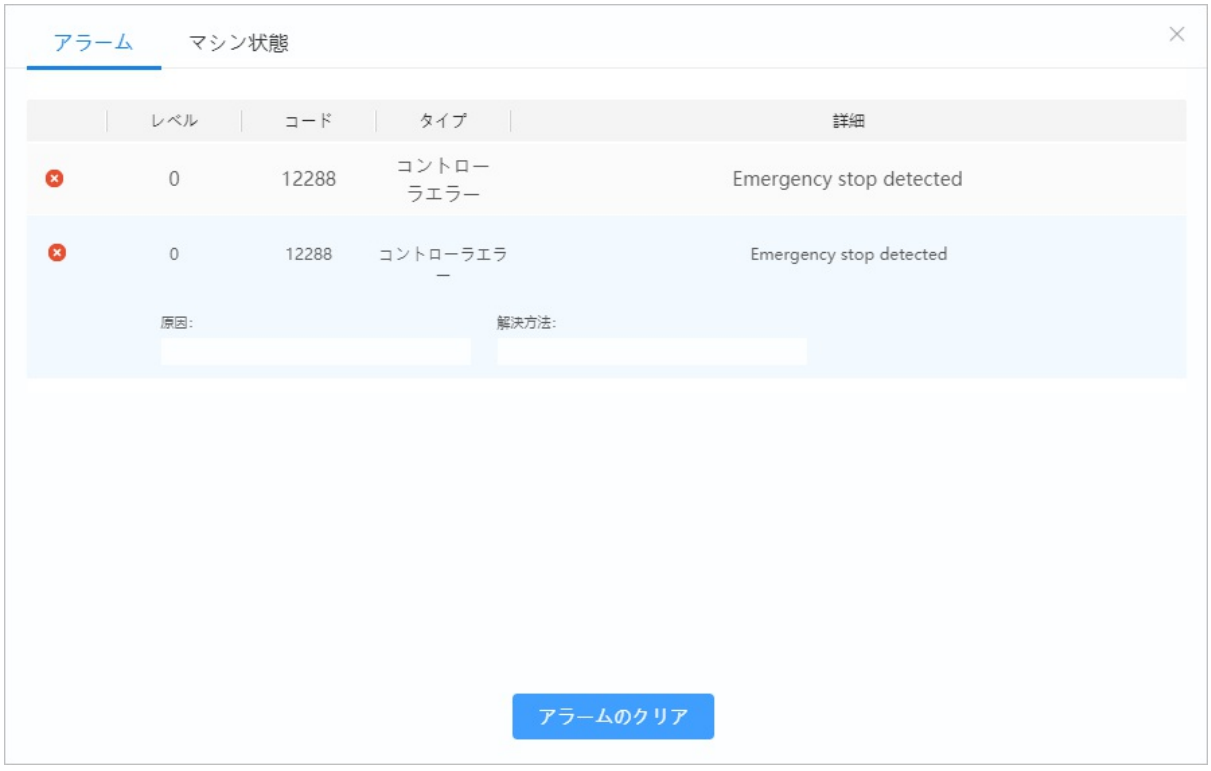
3.2 トップツールバー

番号	説明
1	クリックすると下記のメニューが表示されます。 • 設定：クリックして設定画面を開きます。 • 言語：制御ソフトウェアの表示言語を設定します。 • ヘルプ：ヘルプドキュメント、デバッグツールを含む一連のヘルプ機能を提供します。 • 更新確認：ソフトウェアバージョンの情報と、新しいバージョンのリリース有無を確認します。 • 本ソフトウェアについて：ソフトウェアのコンポーネントに関する説明を確認します。
2	クリックすると、その他画面からメイン画面に戻ります。
3	ロボット接続パネルの詳細は、ロボット接続を参照してください。
4	アラームログボタンの詳細は、以下のアラームログを参照してください。
5	イネーブルボタンの詳細は、以下のイネーブル状態の説明を参照してください。
6	青い点をドラッグするか、速度バーの任意の位置をクリックして、グローバルレートを調整します。グローバルレートとは、ロボットアームの実際の移動速度の計算係数です。速度の計算方法の詳細は、移動パラメータ設定をご覧ください。
7	緊急停止ボタンは、ロボットアームが動作中に突発的な状況が発生した場合に、このボタンを押します。ロボットアームは緊急停止して電源が切ります。詳細は以下の緊急停止ボタンの説明を参照してください。

アラームログ

ジョグまたは点保存の方法が不適切であるか、ロボットアームの使用が不適切である場合、例えばロボットアームの位置制限であったり、特異点にある場合には、アラームがトリガーされます。ロボットアーム実行中にアラームがトリガーされた場合、DobotStudio Proの画面のアラームアイコンは  に変わります。この時にアラームアイコンをクリックすると、アラーム画面が開き、アラーム情報を確認できます。

この時、ユーザーはアラーム情報をクリックして、以下に示されているように、アラームの原因と解決方法を確認し、「アラーム削除」をクリックします。



イネーブル状態の説明

ロボットアームはイネーブル状態になってから動作することができます。

- イネーブルボタンが左側にあり、ロボットアームのアイコンが青色  である場合、ロボットアームはイネーブルオフ状態にあることを示しています。ボタンをクリックすると、角度確認と負荷設定ウィンドウが次々と表示されます（末端の負荷の偏心座標はJ6軸が0°の時に設定してください。負荷重量（末端の治具重量を含む）はCRロボットアームが許容する最大負荷重量を超えないようにしてください）。確認後、ロボットアームはイネーブルオンになります。現時点では、ロボットアームは小さな運動を開始します。その後、末端の指示ランプは緑色で点灯します。これは、ロボットアームがイネーブルオンになっていることを示しています。同時に、イネーブルボタンを右側に移動し、アイコンも緑色  に変わります（右下の黄色の感嘆符は、設定した負荷が0kgであることを示しています）。

> ⚠注意：
>
> 負荷設定が不正確であると、衝突検出異常アラームが発生したり、ドラッグした時に本体が制御できなくなる場合があります。

- イネーブルボタンのアイコンが緑色である安倍、ロボットアームはイネーブル状態にあることを示しています。ボタンをクリックすると、イネーブルオフの確認ボックスが表示されます。確認後、ロボットアームはイネーブルオフオフになります。現時点では、ロボットアーム末端の指示ランプは青で点灯します。これは、ロボットアームがイネーブルオフオフになっていることを示しています。同時に、イネーブルボタンのアイコンも青色に変わります。
- イネーブルボタンが青色で点滅している場合、ロボットアームはドラッグモードにあることを示しています。この時はイネーブルオフオフにしたり、ソフトウェア制御によりロボットアームの移動を制御することができません（プロジェクト実行、ジョグ、RunToによる姿勢指定など）。

緊急停止ボタンの説明

緊急停止ボタンを押すと、ロボットアームは緊急停止して電源が切ります。ボタンアイコンは赤色に変わります。

再度ロボットアームをイネーブルにする場合、まずアラームをクリアし（ポップアップウィンドウのプロンプトに従って、ソフト緊急停止ボタンをリセットする）、電源を入れてイネーブルにします。

>  説明
> 実際の緊急停止ボタンを押した場合、ソフトウェアの緊急停止ボタンのアイコンは変化しません。アラームクリア前にまず、実際の緊急停止ボタンをリセットします（一般的には時計回りに回すことでボタンをリセットする）。

3.3 制御パネル

番号	説明
1	クリックするとパネルを非表示にすることができます。非表示にした後、右側ツールバーの「制御」ボタンをクリックすると再度表示されます。
2	長押しすると、ロボットアームは初期のポーズに移動します。初期のポーズは設定をサポートします。
3	クリックすると、設定ページを開くことができます。
4	クリックすると、制御パネルを折りたたんで表示することができます。折りたたまれた状態で再度クリックすると、折りたたみをキャンセルできます。
5	上にあるユーザー座標系またはツール座標系右側のプルダウンメニューをクリックすると、対応する座標系番号を選択できます（プルダウンメニュー中の「...」をクリックすると、対応する座標系の設定画面を開くことができる）。さらに、下でどの座標系を使用するかを選択します。

6	ユーザーがロボットアームをジョグまたは移動している場合、シミュレーション画面でロボットアームの移動状況をリアルタイムで確認することができます。
7	ロボットアームの移動方式を選択します。 • ジョグ：ユーザーがジョグボタンを押すと、ロボットは継続して移動し、離すと移動を停止します。 • インチング移動：ジョグボタンをクリックすると、インチングで移動します。長押しすると移動を継続できます。デカルト座標系では、インチング動作値の単位はmmです。0.1は、クリックするごとに0.1mm移動するということを意味します。関節座標系では、インチング動作値の単位は°です。0.1は、クリックするごとに0.1°移動するということを意味します。
8	操作パネルをジョグします。左の列は、デカルト座標系のジョグ制御です。右の列は関節座標系のジョグ制御です。 • デカルト座標系で「X+」、「X-」を例に挙げると、「X+」、「X-」ボタンをクリックすると、CRロボットアームをX軸に沿って正または負の方向に移動します。以下同様です。 • 関節座標系で「J1+」、「J1-」を例に挙げると、「J1+」、「J1-」ボタンをクリックすると、CRロボットアームをJ1軸に沿って正または負の方向に回転します。以下同様です。

4 設定

4.1 基本設定

基本設定ページは設備の規格を表示し、ロボットアームの姿勢を設定するために使用されます。

- 「ソフトウェアが次に起動時にデフォルトで自動で接続」をチェックした場合、次のソフトウェアの起動時に現在接続しているロボットアームの接続を自動で試みます。
- マウスを「詳細を表示」に移動し、そこにカーソルを合わせると、詳細のファームウェアバージョン情報を確認できます。

初期の姿勢

初期の姿勢はカスタマイズ可能な姿勢で、そのデフォルトの姿勢は0点ポーズで、即ち各関節の角度がすべて0となっています。

「初期点位置を再設定」をクリックすると、初期の姿勢を変更できます。

各関節の角度を手動で入力、またはロボットアームを指定の姿勢に動かした後、「現在の点位置を取得」をクリックすると、ロボットアームの現在の各関節の角度を読み取ることができます。各関節の角度を確認後、「確認」をクリックすると初期の姿勢を更新できます。

「初期の点位置に移動」を長押しすると、ロボットアームを初期の姿勢に動かせることができます。「デフォルトの点位置に戻る」をクリックすると、初期の姿勢をデフォルトの姿勢に戻すことができます。

梱包姿勢と**0**点の姿勢

梱包姿勢はロボットが占用する空間を小さくし、梱包搬送を容易にすることができます。0点の姿勢の各関節の角度はすべて0度です。「この位置まで移動」を長押しすると、ロボットアームを対応する姿勢まで動かせることができます。

4.2 通信設定

IP設定

ロボットはLANインターフェスを介して外部設備と通信することができます。TCP、UDPまたはModbusプロトコルに対応します。ユーザーはロボットのIPアドレス、サブネットマスクとゲートウェイを変更することができます。外部設備と接続するとき、IPアドレスは外部設備のIPアドレスと同じネットワークセグメントにあり、かつ競合しない必要があります。

コントロールキャビネットが大型コントロールキャビネット（CC162）であるとき、1つのLANインターフェスしかありません。変更するのはこのインターフェスのアドレスです。コントロールキャビネットが小型コントロールキャビネット（CCBOX）であるとき、2つのLANインターフェスがあり、変更するのはLAN1インターフェスのアドレスです。2種類のコントロールキャビネットのLAN1インターフェスのデフォルトIPアドレスはいずれも192.168.5.1です。

- ロボットアームを外部設備と直接接続またはスイッチを介して接続する場合、「IPの手動設定」をチェックし、IPアドレス、サブネットマスクおよびデフォルトのゲートウェイを変更した後、「適用」をクリックします。
- ロボットアームを外部設備とルーターを経由して接続する場合、「IPの自動取得」（ルーターによりIPアドレスを自動割り当て）をチェックして、その後、「適用」をクリックします。

Wi-Fiの設定

ロボットアームはWi-Fiを介して外部設備と通信できます。ユーザーはWi-Fiの名称とパスワードを変更することができ、設備を再起動した後、有効になります。Wi-Fiの初期パスワードは1234567890です。


WIFI設定
WIFI接続用の上位マシンソフトウェア
SSID
Dobot_WIFI
10/20
password
適応

4.3 座標系の管理

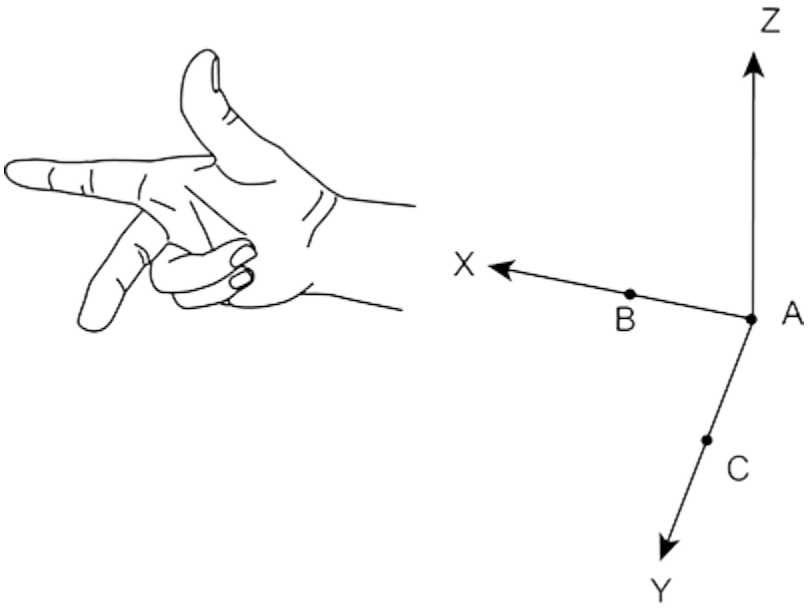
4.3.1 ユーザー座標系

ワークの位置が変化したとき、またはロボットアームの実行プログラムを複数の同じタイプの加工システム内で繰り返し使用する必要があるとき、ユーザー座標系を設定する必要があります。すべての経路がユーザー座標に伴って同期更新されるので、ティーチングプログラミングは大幅に簡単化されます。

現在システムは最大10個のユーザー座標系に対応しています。ユーザー座標系**0**はベース座標系で、変更することができません。

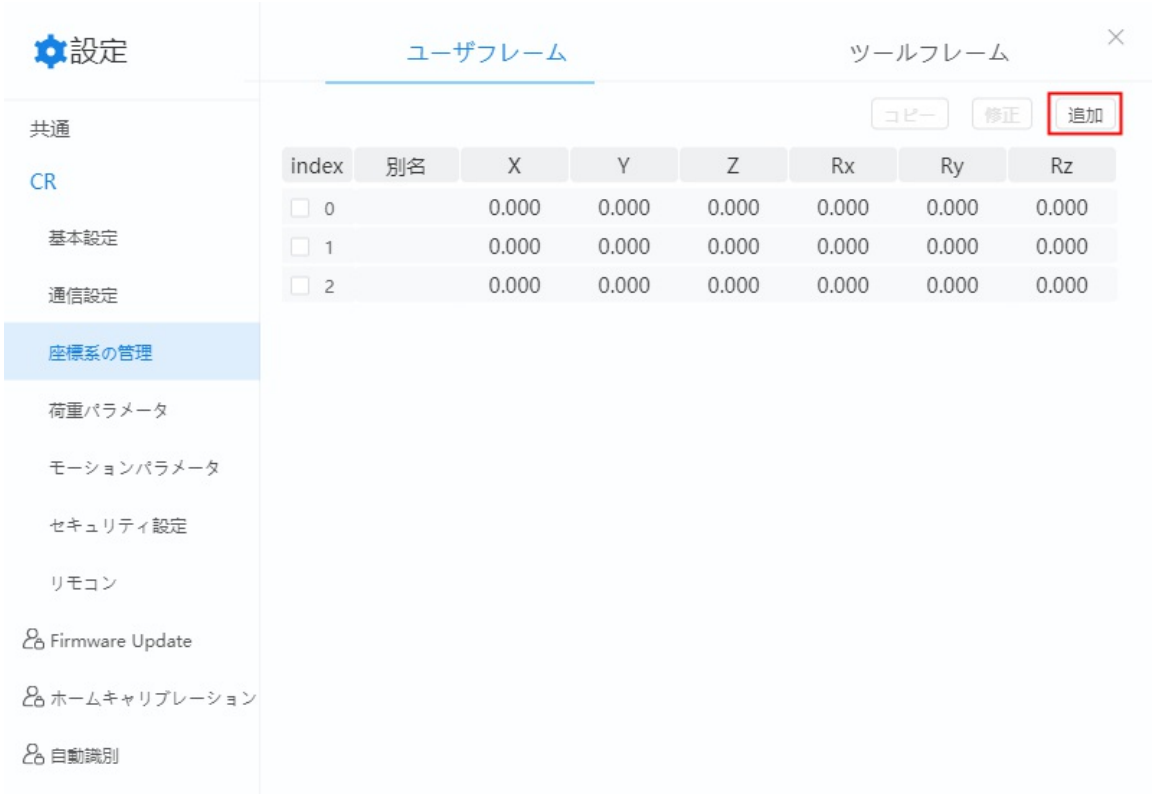
> 説明
>
> ユーザー座標系を確立するとき、基準座標系がベース座標系であることを保証してください。

ユーザー座標系は3点ティーチング法を用いて生成します。ロボットアームを任意の3つの点**A**、**B**と**C**に移動させます。ここで、**A**点を原点とします。**AB**の2点を結ぶ線がユーザー座標系X軸正方向を決めます。点Cに沿ったX軸に垂直な線がY軸正方向を決めます（実際にキャリブレーションするとき、点Cはかならずしも Y軸上になくてもよく、X軸とY軸正方向で決められる平面象限内にあり、かつX軸上になければよく、システムはY軸正方向を自動的に計算することができます）。右手の法則に従って、Z軸方向を決めます。

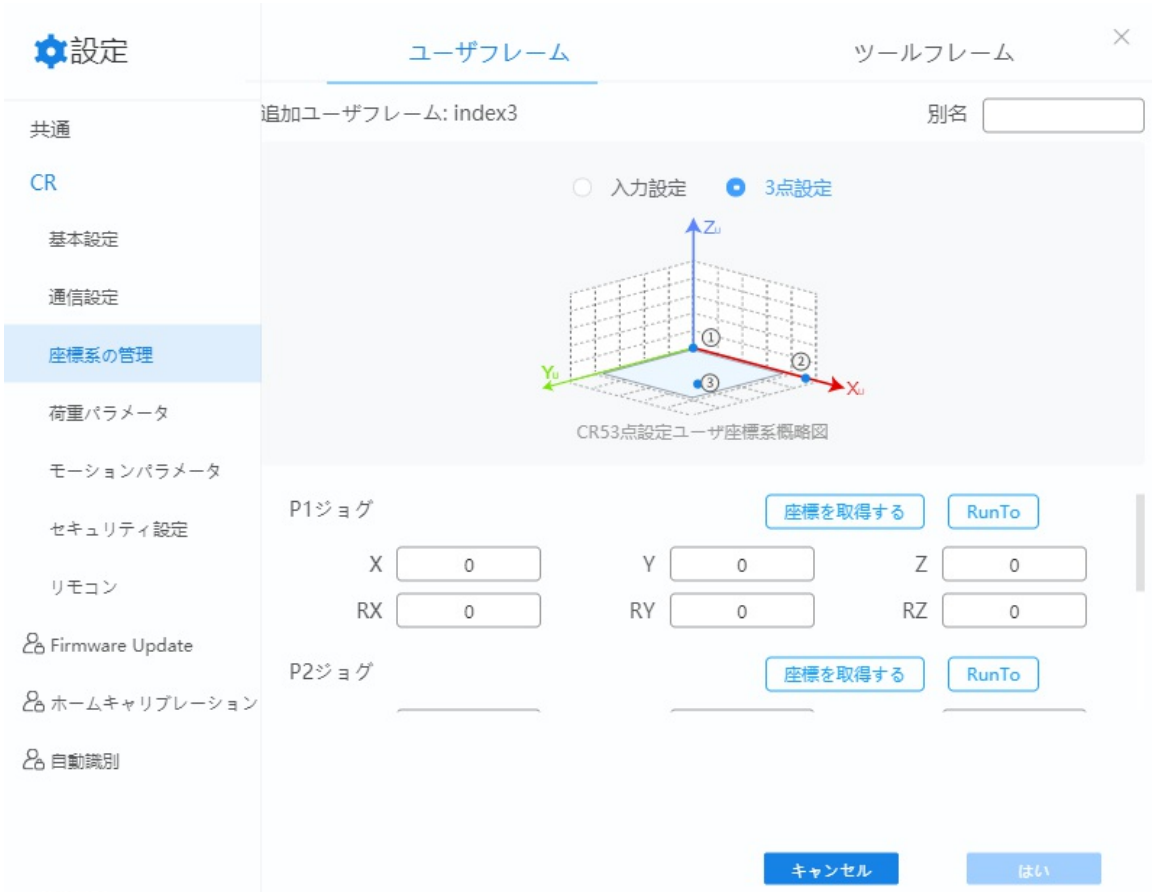


ユーザー座標系の作成

1. 下図に示すとおり、「新規追加」をクリックします。



2. ユーザー座標系の新規追加ページで「3点設定」を選択します。



説明

- ユーザー座標系を確立するとき、基準座標系がベース座標系であること、すなわち、ロボットアームをジョグさせるときのユーザー座標系が「0」であることを保証してください。
- 「RunTo」を長押しすると、ロボットアームを設定済みの点位置に移動させることができます。

3. ロボットアームを点P1にジョグさせ、P1座標パネルで「座標の取得」をクリックします。
4. ロボットアームを点P2にジョグさせ、P2座標パネルで「座標の取得」をクリックします。
5. ロボットアームを点P3にジョグさせ、P3座標パネルで「座標の取得」をクリックします。
6. 「確認」をクリックすると、ユーザー座標系の作成は完了します。

ユーザー座標系を作成または変更した後に、下図に示すとおり、制御パネルでユーザー座標系を選択してジョグさせます。

説明
ユーザー座標系を新規追加または変更するとき、ユーザーは「設定の入力」を選択するか、X、Y、Z、Rx、Ry、Rzの値を直接変更してから、「確認」をクリックすることができます。

その他の操作

- 座標系のチェックボックスを選択した後に、「変更」をクリックすると、作成済みの座標系を変更することができます。変更方法は作成方法と同様であり、ここでは説明を省略します。
- 座標系のチェックボックスを選択した後に、「複製」をクリックすると、選択した座標系と同じの新しい座標系を作成することができます。

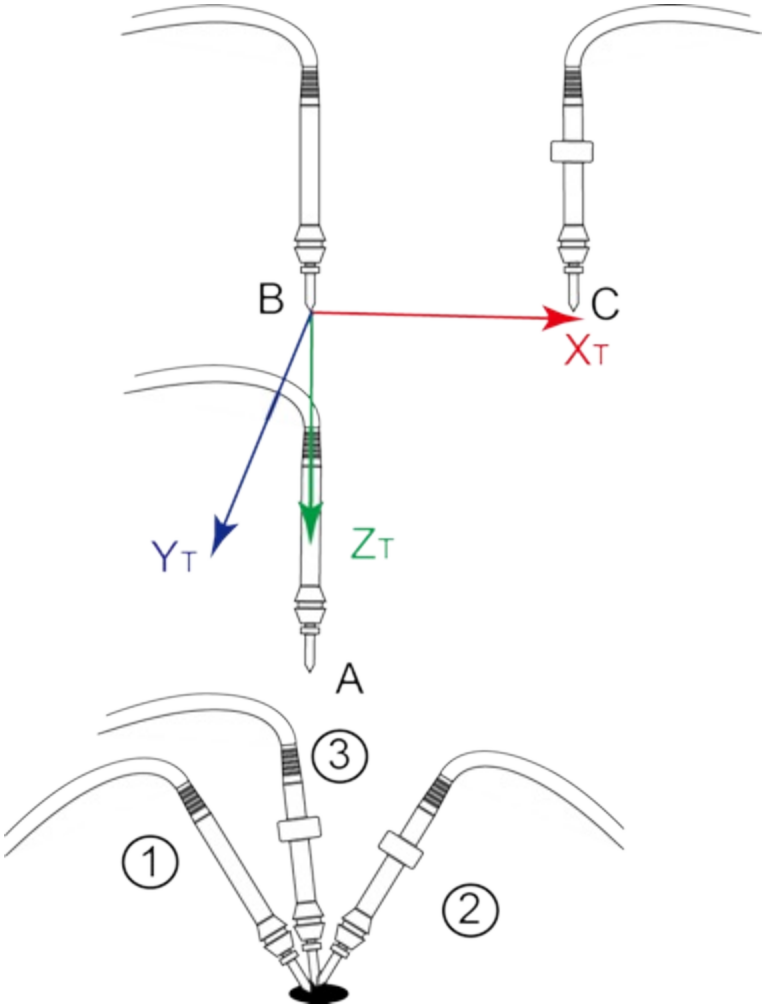
4.3.2 ツール座標系

ロボットアームの先端にツール（例えば、溶接ガン、ノズル、クランプなど）を取り付けた後、プログラミングおよびロボットアームの動作のため、このときにツール座標系を設定する必要があります。例えば、複数のクランプを使って複数のワークを同時に運搬する際には、運搬効率を向上させるために各クランプを独立したツール座標系に設定することができます。

現在のシステムは最大10個のツール座標に対応しています。ツール座標系**0**はデフォルトツール座標系で、すなわちツールを使用せず、変更もできません。

>  説明
>
> ユーザー座標系を確立するとき、基準座標系がベース座標系であることを保証してください。

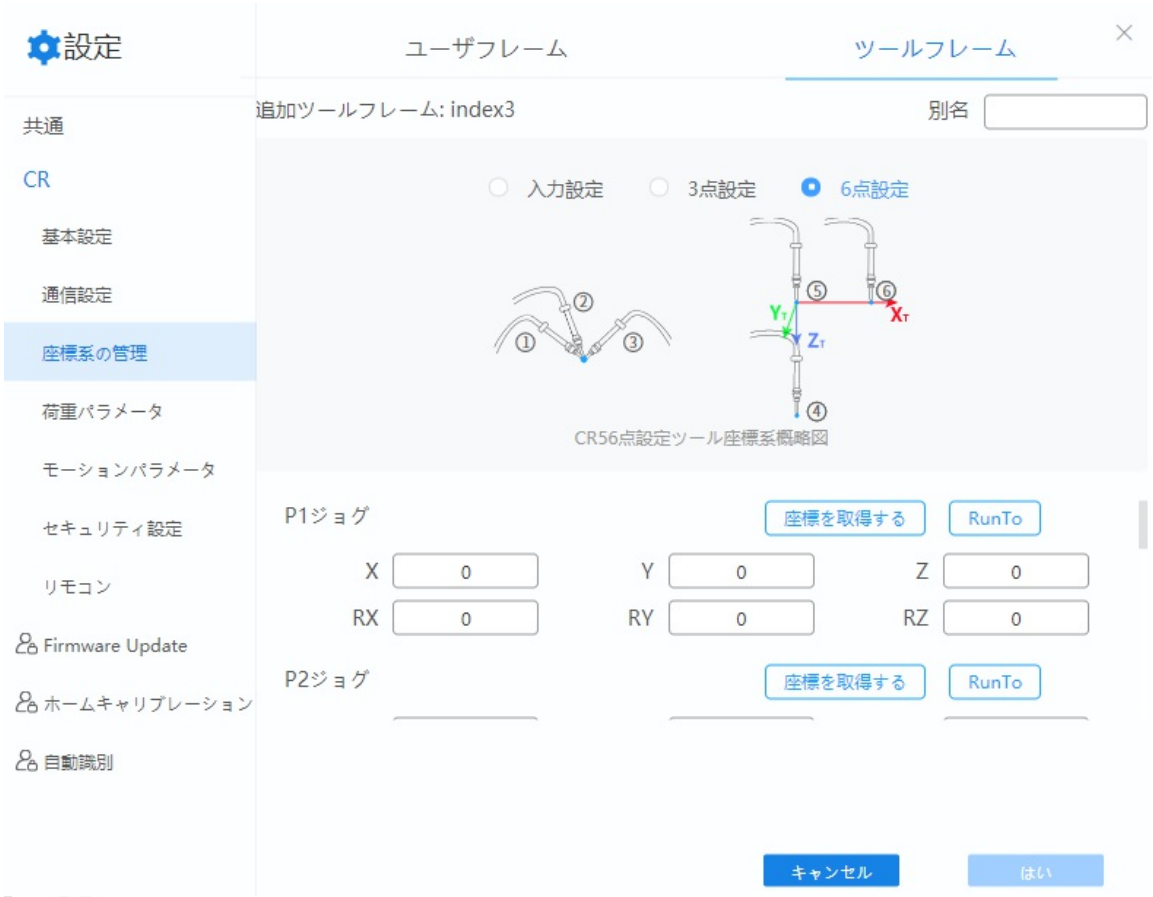
6軸ツール座標系は3点ティーチング法「TCP+ZX」を用いて生成します。ロボットアームの先端にツールを取り付けた後、TCP（Tool Center Point）を3つの異なる方向で空間中の同一の点（基準点）に合わせて、ツールの位置オフセットを取得します。別の3つの点（**A**、**B**、**C**）に基づいてツール姿勢オフセットを取得します。



ツール座標系の作成

1. ロボットアームの先端にツールを取り付けることに関しては、本節で詳細に説明しません。
2. 下図に示すとおり、「新規追加」をクリックします。

3. ツール座標系の新規追加ページで「6点設定」を選択します。



説明

- ツール座標系を確立するとき、基準座標系がベース座標系であること、すなわち、ロボットアームをジョグさせるときのツール座標系が「0」であることを保証してください。
- 「RunTo」を長押しすると、ロボットアームを設定済みの点位置に移動させることができます。

4. ロボットアームをジョグさせ、TCPを第1のポーズ（概略図のポーズ①）で基準点に合わせ、P1座標パネルで「座標の取得」をクリックします。
5. ロボットアームをジョグさせ、TCPを第2のポーズ（概略図のポーズ②）で基準点に合わせ、P2座標パネルで「座標の取得」をクリックします。
6. ロボットアームをジョグさせ、TCPを第3のポーズ（概略図のポーズ③）で基準点に合わせ、P3座標パネルで「座標の取得」をクリックします。
7. ロボットアームをジョグさせ、TCPを垂直ポーズ（概略図のポーズ④）で基準点に合わせ、P4座標パネルで「座標の取得」をクリックします。
8. Z軸を正方向にジョグさせ、ロボットアームを他の点（概略図のポーズ⑤）に移動させ、P5座標パネルで「座標の取得」をクリックします。
9. X軸を正方向にジョグさせ、ロボットアームを点P6（概略図のポーズ⑥。2つの点P4、P5と同一直線上にあってはならない）に移動させ、P6座標パネルで「座標の取得」をクリックします。
10. 「確認」をクリックすると、ツール座標系の作成は完了します。

ツール座標系を作成または変更した後に、下図に示すとおり、制御パネルでツール座標系を選択してジョグさせます。

説明

ツール座標系を新規追加または変更するとき、ユーザーは実際の状況に応じて「設定の入力」または「3点設定」を選択することができます。

その他の操作

- 座標系のチェックボックスを選択した後に、「変更」をクリックすると、作成済みの座標系を変更することができます。変更方法は作成方法と同様であり、ここでは説明を省略します。
- 座標系のチェックボックスを選択した後に、「複製」をクリックすると、選択した座標系と同じの新しい座標系を作成することができます。

4.4 負荷パラメータの設定

ロボットアーム自体が持つ性能を十分に発揮するため、先端負荷質量と偏心座標を許容範囲内に設定し、ロボットが動作する場合にJ6軸が偏心しないことを確保してください。合理的な設定により、ロボットアームの移動は最適化され、振動を抑制し、作業時間を短縮できます。

> 説明：
> ロボットアームをイネーブルオンにする場合には必ず、負荷パラメータの設定を求めるウィンドウがポップアップ表示され、設定したパラメータをその画面に同期します。

「変更」ボタンをクリックすると、負荷パラメータを変更できます。

- 先端負荷の偏心座標はJ6軸が0°の時に設定してください。
- 先端負荷重量は、治具の先端重量とワーク重量の和で、ロボットアームが許容する最大負荷を超えないようにしてください。

> 注意：
>
> 負荷設定が不正確であると、衝突検出異常アラームが発生したり、ドラッグした時に本体が制御できなくなる場合があります。

設定を変更後、「確認」ボタンをクリックすると有効になります。

適切な負荷質量と偏心座標を設定してください。そうでない場合、エラーまたはショックが発生し、機器の耐用年数が短くなります。

4.5 移動パラメータ設定

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

ジョグ設定

実際の状況に応じて、ジョグ時の関節または座標系の各軸の最大速度、最大加速度を設定し、設定完了後に「保存」をクリックしてください。

ロボットの実際の移動速度/加速度 = ここで設定した速度/加速度 x グローバルレート。

例：関節速度を12°/sとし、グローバルレートを50%とすると、実際のジョグ速度は12°/s x 50% = 6°/sとなります。

をクリックすると、対応するセクションのすべての設定をデフォルト値に復元します。

再現設定

実際の必要に応じて、関節または座標系における各軸の移動速度、加速度、加加速度を設定し、設定完了後、「保存」をクリックしてください。

ロボットの実際の移動速度/加速度 = ここで設定した速度/加速度 x グローバルレート x プログラミング時速度インストラクションで設定した百分率。

例：座標系速度を2000mm/sとし、グローバルレートを50%とし、プログラミング時の移動インストラクションに設定した速度を80%とすると、実際の移動速度は2000mm/s x 50% x 80%= 800mm/sとなります。

をクリックすると、対応するセクションのすべての設定をデフォルト値に復元します。

4.6 安全設定

4.6.1 衝突検出

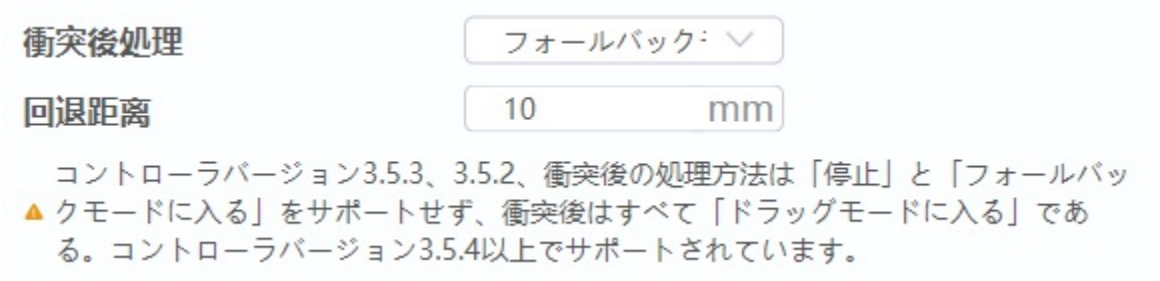
衝突安全機能は、主にロボット本体への衝突力の影響を軽減し、ロボット本体や周辺設備の損傷を回避するために使用されます。衝突安全機能をオンにした後、ロボットアームがワークまたはその他の障害物に衝突した時、機器またはオペレーターの衝突による損傷（怪我）を防ぐため、ロボットアームは動作を自動的に停止します。

衝突検出の敏感度が高いほどロボットアームの衝突検出に必要な力が小さくなります。トップツールバーのロボット接続パネルに現在の安全レベルが表示されます。

ロボットアームのジョグ中、停止に必要な力が検出されると、ポップアップウィンドウが表示されます。この時、衝突の原因を解決し、「リセット」をクリックする必要があります。衝突の原因を解決するために制御ソフトウェアを操作する必要がある場合、「1分後通知する」をクリックして、ポップアップウィンドウを一時的に閉じることができます（1分後、再びポップアップウィンドウで通知する）。

「衝突後の処理方法」はロボットアームのプロジェクト実行中に衝突が発生した後の処理方法を表します。

- 停止：ロボットアームがプロジェクト実行を停止します。
- 一時停止：ロボットアームの一時停止には、実際の状況に応じて衝突の原因を解決してプロジェクトの実行を再開するか、プロジェクトの実行を停止するか選択する必要があります。
- ドラッグモードに進む：ロボットアームはプロジェクトの実行を停止し、ドラッグモードに自動的に進みます。
- フォールバックモードに進む：ロボットアームは衝突前の軌跡に基づき指定の距離を自動でフォールバックし、フォールバック距離の設定範囲は0～50mmです。



4.6.2 関節ブレーキ

ロボットアームがイネーブルオフ状態にある時、関節の移動を防止するため、関節は自動的にブレーキをかけてモーターの位置ロックを保持し、ロボットアームの可動部分が自重または外力により移動しないよう確保します。

ユーザーが関節ドラッグ操作を行う必要がある場合、ブレーキをオンにできます。即ち、ロボットアームをイネーブルオフにした後、手動で関節を支え、その後、移動する関節に対応するスイッチをクリックします。

> ⚠ 注意
>
> ブレーキをオンにする時、ロボットアームが自重で下に落ちることを回避するため、手動で関節を支える必要があります。

4.6.3 取り付け設定

一般的な状況において、ロボットアームは安定した台または地面に設置します。このような状況では、このページでいかなる操作も行う必要はありません。ロボットアームが吊り掛け式、壁掛け式の取り付け、あるいは一定角度のある取り付けを採用する場合、イネーブルオフ状態において回転角と傾斜角を設定してください。

ユーザーは手動または自動の2種類の方法でキャリブレーションすることができます。

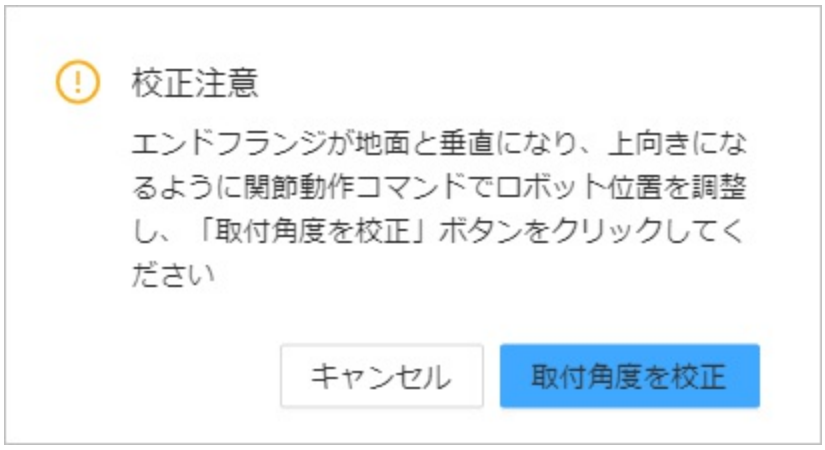
手動キャリブレーション

実際の取り付け姿勢に基づき、ページ左側で対応する取り付け姿勢を選択するか、「カスタム」を選択した後、下で傾斜角と回転角を調整します。

- 傾斜角とは、本体がホームポジションでX軸を中心に反時計回りに回転する角度のことです。
- 回転角とは、本体がホームポジションでZ軸を中心に反時計回りに回転する角度のことです。

自動キャリブレーション

ロボットアームを設置し、イネーブルにした後、「キャリブレーション」をクリックし、表示されたダイアログボックスに従って操作を行うことにより、傾斜角と回転角を得ることができます。



キャリブレーション完了後、「保存」をクリックすると、設定が有効になります。

「デフォルトに戻す」をクリックすると、キャリブレーションした角度をデフォルト値に戻すことができます。

4.6.4 感度設定

感度設定は主にロボットアームの各関節のドラッグ操作における感度の度合を調節するために使用されます。

感度が低いほど、ドラッグ中の抵抗が大きくなります。

4.6.5 高度な機能

現場の状況から高度な機能をオンオフにする必要がある時、この画面で構成できます。特に要求がない場合、デフォルト値を維持することをお勧めします。

> 説明
>
> 高度な機能は管理者パスワード（デフォルトのパスワード：888888）を入力後に使用可能となります。

4.7 電源電圧（CCBOX）

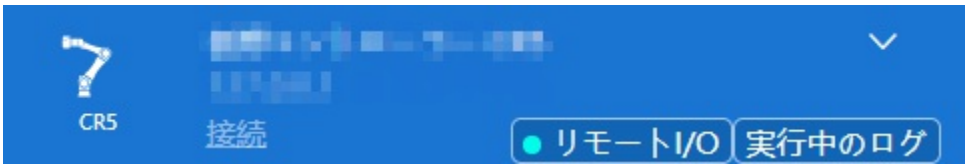
コントロールキャビネットのタイプがCCBOX（小型コントロールキャビネット）であるとき、コントロールキャビネットの入力電源電圧範囲を設定する必要があります。

入力電圧の実際の範囲に基づいて設定してください。最小値は30V以上、最大値は60V以下とします。

4.8 リモート制御

外部設備は、例えば、リモートI/Oモード、リモートModbusモードなど異なるリモート制御モードでインストラクションを発行してロボットアームを制御することができます（すなわち、ティーチング済みのプログラムファイルに対して実行の制御を行う）。

> 説明
>
> - リモート制御モードの切り替えは、ロボット制御システムの再起動を必要としません。
> - ロボット制御システムのモードに関係なく、非常停止スイッチはいつも有効です。
> - ロボットアームがリモート制御モードで動作している場合、このときに他の動作モードに切り替えるとプロジェクトは自動的に停止します。
> - ロボットの電源を投入して初期化する前に制御信号を送信しないでください。ロボットが異常動作するおそれがあります。
> - リモート制御モードに入ると、トップツールバーに現在のモードが表示されます。右側の「実行ログ」をクリックすると、リモート制御の実行ログを表示することができます。下図はリモートI/Oモードを例とします。



オンラインモード

デフォルトの制御モード。DobotStudio Proを使用してロボットアームに対して制御を行います。

リモートModbus

リモート制御モードがリモートModbusであるとき、外部設備はリモートModbusを介してロボットアームを制御することができます。

Modbusレジスタの特定の機能は上図に示すとおりです。「クリックして変更する」をクリックして編集することができます。

リモートModbusモードでプロジェクトを実行するステップは以下のとおりです。

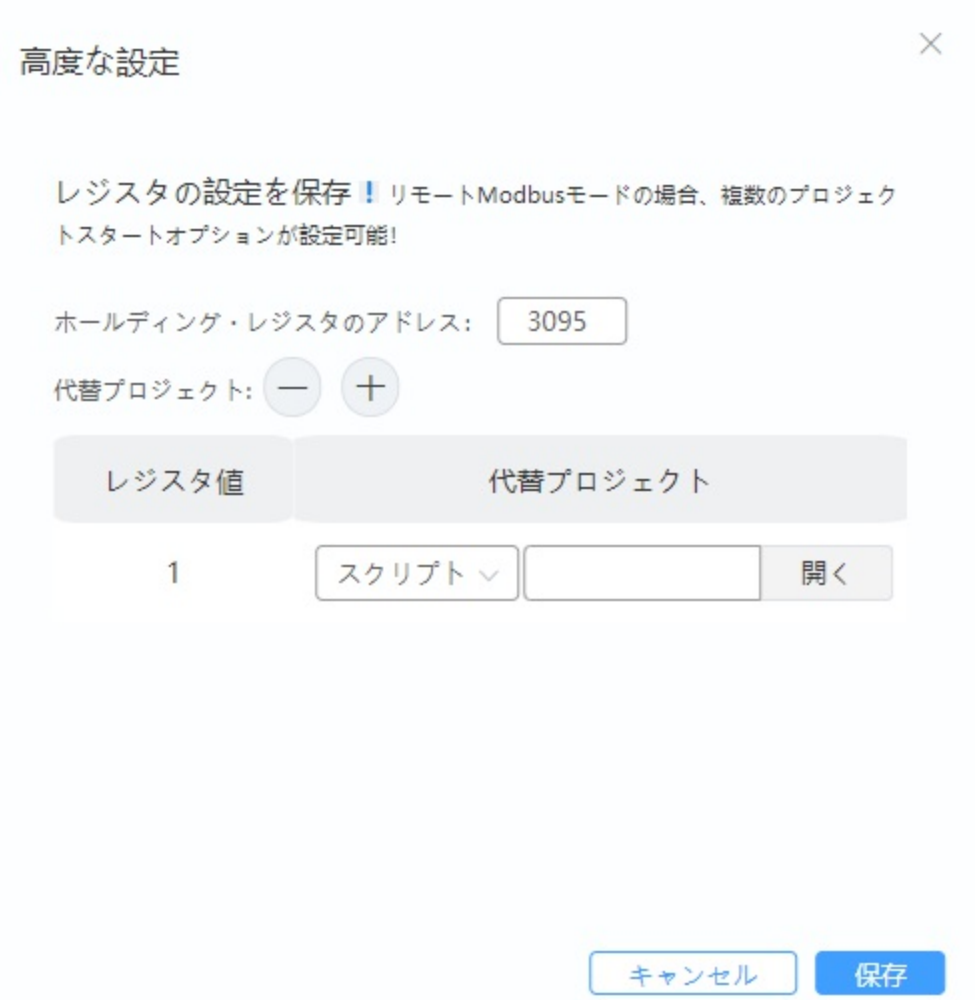
前提条件

- リモート実行用のプロジェクトファイルを準備しており、かつ正確に実行できること。
- LANインターフェスを介して外部設備とロボットアームを接続したこと。直接接続するか、またはルーターを介して接続してもよく、実際の状況に応じて選択してください。ここで、ロボットアームと外部設備のIPアドレスは同じネットワークセグメントにある必要があります。ロボットアームのデフォルトIPアドレスは192.168.5.1です。通信設定で設定することができます。
- ロボットアームが電源オンであること。

操作手順

1. 「現在のモード」として「リモートmodbusモード」を選択し、リモート実行待ちのプロジェクトを選択します。グラフィカルプログラミングまたはスクリプトプログラミングのプロジェクトに対応します。

2. Modbusを介して複数の異なるプロジェクトを起動する必要がある場合、「詳細設定」をクリックすることができます。詳細設定で、下図に示すとおり、選択候補プロジェクトの保存レジスタアドレスを設定し、および選択候補プロジェクト一覧を構成することができます。



3. 「適用」ボタンをクリックすると、このとき、ロボットアームはリモートModbus制御モードに入ります。制御ソフトウェア非常停止以外のインストラクションを受け付けません。
4. 外部設備で起動信号をトリガーすると、このとき、ロボットアームは選択したプロジェクトファイルに従って動作を行います。
5. 停止信号をトリガーすると、移動を停止します。

リモートI/O

リモート制御モードがリモートI/Oであるとき、外部設備はリモートI/Oを介してロボットアームを制御することができます。

制御システムの特定のI/Oポートの定義は上図に示すとおりです。「変更」をクリックして編集することができます。

コントロールキャビネットタイプがCCBOX（小型コントロールキャビネット）であるとき、安全I/Oと汎用I/Oは端子を共用します。安全I/Oに構成された端子はリモートI/Oに構成することはできません。

「詳細設定」をクリックして、グループI/Oの方式で複数の選択候補プロジェクトを構成することができます。

1. 「+」または「-」をクリックすると、グループI/Oに割り当てられるアドレスの数量を上げたり下げたりすることができます。割り当てられるアドレスが多いほど、構成できる選択候補のプロジェクトが多くなります。

   - 0個のアドレス：選択候補のプロジェクトを構成できません。
   - 1個のアドレス：1個の選択候補のプロジェクトを構成できます。
   - 2個のアドレス：3個の選択候補のプロジェクトを構成できます。
   - 3個のアドレス：7個の選択候補のプロジェクトを構成できます。
   - 4個のアドレス：15個の選択候補のプロジェクトを構成できます。
2. ドロップダウンリストを使用して割り当てられるアドレスを変更することができます。グループI/Oに割り当てられるアドレスはリモートI/Oまたは安全I/O（CCBOX）と重複してはなりません。

3. アドレスを割り当てた後に、各グループI/Oの値に対して選択候補プロジェクトを設定することができます。少なくとも1つを設定します。

   - リモートI/Oモードで、プロジェクトを実行する前に、対応するグループI/O値（青色のハイライトはONを表し、グレイはOFFを表す）を設定することにより、対応する選択候補プロジェクトを選択します。

   - 上図のDI1~DI4の4個のアドレスの割り当てを例とします。

DI1がONで、DI2、DI3、DI4がOFFであることは、第1の選択候補プロジェクトを選択することを表します。

DI1和DI2がONで、DI3、DI4がOFFであることは、第3の選択候補プロジェクトを選択することを表します。

DI1~DI4がいずれもONであることは、第15の選択候補プロジェクトを選択することを表します。

- グループI/OがすべてOFFに設定されているとき、上位階層のページで構成したメインプロジェクトを選択することを表します。

4. 「保存」をクリックして構成を完了します。

リモートI/Oモードでプロジェクトを実行するステップは以下のとおりです。

前提条件

- リモート実行用のプロジェクトファイルを準備しており、かつ正確に実行できること。
- デジタルI/Oポートを介して外部設備とロボットアームを接続したこと。
- ロボットアームが電源オンであること。

操作手順

1. 「現在のモード」として「リモートI/Oモード」を選択し、リモート実行待ちのプロジェクトを選択します。グラフィカルプログラミングまたはスクリプトプログラミングのプロジェクトに対応します。
2. 「適用」ボタンをクリックすると、このとき、ロボットアームはリモートI/O制御モードに入ります。制御ソフトウェア非常停止以外のインストラクションを受け付けません。
3. （オプション）外部設備でグループI/Oをトリガーして実行したいプロジェクトを選択します。
4. 起動信号をトリガーします。このとき、ロボットアームは選択したプロジェクトファイルに従って動作を行います。
5. 停止信号をトリガーすると、移動を停止します。

TCP/IP二次開発

このモードは顧客がTCPに基づいて制御ソフトウェア自主開発を行う場合にのみ使用されます。制御ソフトウェアを自主開発する必要がある場合には、（Githubでホストされている）Dobot TCP/IPプロトコルを参照してください。

4.9 0点キャリブレーション

ロボットアームのモーター、減速機などのトランスミッション部品を交換する際、またはワークと衝突したときなど状況では、ロボットアームの0点位置に変化が生じます。このときロボットアームに対して0点キャリブレーションを行い必要があります。

> 説明
>
> - ホームキャリブレーションは、ホームポジションが変更された場合にのみ使用されます。慎重に操作してください。
> - 0点キャリブレーション機能は管理者パスワード（デフォルトパスワード：888888）を入力しなければ使用することができません。

ページのプロンプトに従って、ロボットアームを0点姿勢に移動させます（ロボットアームの各関節の0点ステッカを使用してキャリブレーションすることができます。詳細についてはハードウェアマニュアルを参照してください）。それからイネーブル状態で「0点キャリブレーション」ボタンをクリックします。

キャリブレーションが正常に完了すると、制御パネルで関節座標を表示します。このとき、J1~J6の値はいずれも0になっています。

4.10 メーカー機能

メーカ機能には、自動識別機能と衝突検出スイッチが含まれています。不適切な操作をすると、ロボットアームの異常または安全のリスクにつながる可能性がありますので、技術サポート指導の下で使用してください。本マニュアルでは紹介しません。

5 I/Oモニタリング

I/Oモニタリングページは、コントロールキャビネットおよび本体先端の各I/O状態をモニタリングし、設定することができます。各IOの具体的な定義については、対応する型番のロボットアームハードウェアハンドブック中のIOインターフェスの説明を参照してください。それぞれのコントロールキャビネットのIO数は異なります。本文中の図は参考です。

番号	説明
1	クリックするとパネルを非表示にすることができます。非表示にした後、右側ツールバーの「I/O」ボタンをクリックすると再度表示されます。
2	クリックすると拡張I/Oを追加できます。Modbus通信のI/Oをモニタリングできます。詳細は以下のI/O拡張を参照してください。

3	クリックすると、I/Oの別名を設定し、表示するかどうかなどを設定できます。詳細は以下のI/Oの構成を参照してください。
4	クリックすると、コントロールパネルを折りたたんで表示することができます。折りたたまれた状態で再度クリックすると、折りたたみをキャンセルできます。
5	I/Oモニタリングエリア。詳細は以下のモニタリングエリアを参照してください。

I/O拡張

- ID：スレーブデバイスのID。
- 名称：I/O拡張の名称。
- IPアドレス：Modbusデバイスのアドレス。
- ポート番号：Modbus通信のポート番号。
- DI/DO：チェックした後にDI/DOを構成するレジスタアドレスと数量。アドレスは0から割り当ててください。

「確定」をクリックした後、モニタリングページの一番下に新しいI/Oが表示されますが、モニタリング機能を有効にするにはコントロールキャビネットの再起動が必要です。

対応するI/Oの右側にあるxをクリックすると、その拡張I/Oを削除できます。

I/Oの構成

- I/Oをチェックすると、モニタリングページに表示されます。
- I/O右側で、そのI/Oの別名を入力できます。別名はモニタリングページに表示されます。同時に、ユーザーはグラフィックプログラミングとスクリプトプログラミング中に、別名により対応するI/Oを呼び出すこともできます。
- コントロールキャビネットのタイプがCCBOXである場合、下図に示されているように、デジタル入力/出力を設定できるタイプはPNP（ハイレベル有効）またはNPN（ローレベル有効）です。

モニタリングエリア

コントロールキャビネットのI/Oと先端のI/Oタブは、以下の機能をサポートします。

- 出力：デジタル出力またはアナログ出力の値を設定します。デジタル出力は、右側の対応するスイッチ状態を直接クリックすると、切り換えることができます。アナログ出力は、まず「変更」ボタンをクリックし、出力値を修正した後、「変更を確定」をクリックすると有効になります。
- モニタリング：入力、出力した実際の状態を確認します。デジタル入力右側の丸の点は対応するDIの状態を示します。グレーの場合はDIがトリガーされておらず、緑はDIがトリガーしたことを示します。
- シミュレーション：デジタル入力状態をシミュレーションし、プログラムのデバッグおよび実行に便利です。対応するDIの状態表示エリアをクリックすると、設定ウィンドウが表示されます。「バーチャル」を選択し、「DI変換」をチェックすると、そのDIはバーチャルトリガー状態（青の丸点）に変わり、ロジック上ではONとみなされます。「DI変換」をチェックしない場合。そのDIは真実の状態を保持します。

コントロールキャビネットのタイプがCC162である場合、アナログ入力/出力は電圧モードまたは電流モードに設定することをサポートしています。「修正」をクリックし、対応するモードを選択します。さらに「修正確認」をクリックすると切り替わります。右側の数値の単位はリアルタイムに変化します。アナログ入力値は確認するだけで、修正はできません。

> 説明
> この機能はハードディップスイッチとあわせて使用します。ディップスイッチはコントロールキャビネット内にあります。設定が必要な場合には、技術サポートまでご連絡ください。

安全I/Oタブは、各安全I/Oポートの機能を設定することができます。安全I/Oポートの詳細は、対応する型番のロボットアームのハードウェアマニュアルの安全I/Oポートの説明を参照してください。

コントロールキャビネットのタイプがCCBOXである場合、安全I/Oと汎用I/Oは端子を共有し、既にリモートI/Oとして構成されている端子は、安全I/Oに構成することはできなくなります。

説明

実際に使用する場合、SI信号の変化間隔をできる限り150ms以上となるように確保してください。そうでない場合、SI信号のジャンプによりロボットの動作状態が異常となる場合があります。

例えば、保護停止リセット入力を設定していない場合、保護停止入力信号のジャンプ（変化間隔は150ms未満）により、ロボットが一時停止した後、自動で再開しない場合があります。以下の方法で解決してください。

1. 制御ソフトウェアによりプロジェクトを一時停止し、プロジェクトの実行を継続します。
2. 保護停止入力信号が再度起動した場合、150ms以上を保持します。

6 Modbus

Modbus機能は、Modbusマスターとしてモドバスのスレーブに接続するためのものです。

番号	説明
1	クリックするとパネルを非表示にすることができます。非表示にした後、右側ツールバーの「Modbus」ボタンをクリックすると再度表示されます。
2	クリックすると、Modbusスレーブに接続できます。詳細は以下のスレーブ接続を参照してください。
3	クリックすると、制御パネルを折りたたんで表示することができます。折りたたまれた状態で再度クリックすると、折りたたみをキャンセルできます。
4	既に接続されているスレーブのレジスタ情報を表示します。

スレーブ接続

- スレーブIP：Modbusデバイスのアドレス。コントロールキャビネット自体のMoedbusスレーブに接続する場合に入力するコントロールキャビネットのIPアドレス（例えば192.168.5.1）です。
- ポート番号：Modbus通信のポート番号。コントロールキャビネット自体のModbusスレーブに接続する場合に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。コントロールキャビネット自体のModbusスレーブに接続する場合、付属書A　Modbusレジスタの定義を参照してください。
- スキャン回数：ロボットアームがスレーブをスキャンする時間間隔。

7 グローバル変数

この画面はグローバル変数の構成と確認のために使用されます。

グローバル変数設定後、グラフィカルプログラミング中にグローバル変数関連のブロックを使用して呼び出すことができ、スクリプトプログラミング中では直接、変数名により使用することができます。

次のタイプのグローバル変数をサポートします。

- bool：ブール値
- String：文字列
- int：整数
- float：倍精度浮動小数点
- point：ロボットアームの点位置は、下図に示されているように、ロボットアームを指定された点位置に移動することにより取得できます。

グローバル保存：

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

グローバル変数の名前付けの推奨

`g_projectの略語_変数の意味` の形式でグローバル変数の名前を付けることをお勧めします。短すぎる名前の使用は避けてください。

グローバル変数名は、luaの予約キーワードやDobotの定義された関数名や変数名と重複してはならないことに注意してください。

```
--luaの予約キーワード
and, break, do, else, elseif, end, false, for,
function, if, in, local, nil, not, or, repeat,
return, then, true, until, while, goto

--Dobotの定義済みの関数と変数（内部予約関数を含む）
Accel, AccelR, AccelS, AI, AO, AOExecute, Arc3,
Arc3Weave, Arch, BlockHighlight, CalcTool, CalcUser,
CheckGo, CheckMove, Circle3, CnvVison, CollisionLogInfo,
CP, CRC16, CyHelix, DestroyCam, DI, DIGroup, DO,
dobot_hook, DOExecute, DOGroup, ECP, ECPSet, ElapsedTime,
EventLog, FC, FCAct, FCCalib, FCCondDisplac, FCCondDisplace,
FCCondForce, FCCondTCPSpeed, FCCondTorque, FCCondTorque,
FCDeact, FCForce, FCLine, FCLoadID, FCRefDirect, FCRefForce,
FCRefLine, FCRefRot, FCRefSpiral, FCRefTorAdjust, FCRefTorque,
FCRel, FCRot, FCSetCompliance, FCSetGain, FCSpiral, FCTorque,
GetABZ, GetAngle, GetCnvObject, GetCoils, GetHoldRegs,
GetInBits, GetInRegs, GetLayerIndex, GetPartIndex, getPathPose,
```

```
GetPathStartPose, GetPose, GetRunningData, GetTraceStartPose,
Go, GoIO, GoR, GoToolR, GoUserR, GRead, GWrite, IceCreamGO,
IceCreamP, IceCreamR, init(pix), InitCam, InverseSolution,IsDone,
Jerk, JerkR, JerkS, Jump, LimZ, Linear, Linear, LoadSet, LoadSwitch,
Log, LogFormat, MatrixPallet, MD, ModbusClose, ModbusCreate,
Move, MoveIn, MoveIO, MoveJ, MoveJIO, MoveJR, MoveOut, MoveR,
MoveToolR, MoveUserR, MoveWeave, NewPose, OFF, ON, PalletSync,
Pause, print, QuitScript, ReadDB, RecvCam, Release, RelJointMovJ,
RelMovJTool, RelMovJUser , RelMovLTool, RelMovLUser, Reset,
ResetElapsedTime, RJ, Rotation, RP, ScrewCommitErrInfo,
ScrewCommitInfo, ScrewCommitProductInfo, ScrewDriverGetError,
ScrewDriverGetResult, ScrewGetSelectedSN, ScrewSetCurrentSN,
SendCam, ServoJ, ServoP, SetABZPPC, SetBackDistance,
SetCnvPointOffset, SetCnvTimeCompensation, SetCoils, SetCollisionLevel,
SetDOMode, setExcitMode, SetExicitCmd, SetExicitCmdZ, SetHoldRegs,
SetObstacleAvoid, SetPartIndex, SetSafeSkin, SetStackMode,
SetTool, SetTool485, SetToolBaudRate, SetToolPower, SetUnstackMode,
SetUser, SetWeldParams, SixForceHome, Sleep, Sleep(period), Speed,
SpeedFactor, SpeedR, SpeedS, Spiral, StartCamera, StartPath,
StartTrace, StopMove, StopSyncCnv, Sync, SyncCnv, SyncExit, Systime,
TCPCreate, TCPDestroy, TCPRead, TCPReadMultiVision, TCPReadVision,
TCPSpeed, TCPSpeedEnd, TCPStart, TCPWrite, TeachPallet, ToolAI,
ToolAnalogMode, ToolDI, ToolDO, ToolDOExecute, TriggerCam,
UDPCreate, UDPRead, UDPWrite, Wait
```

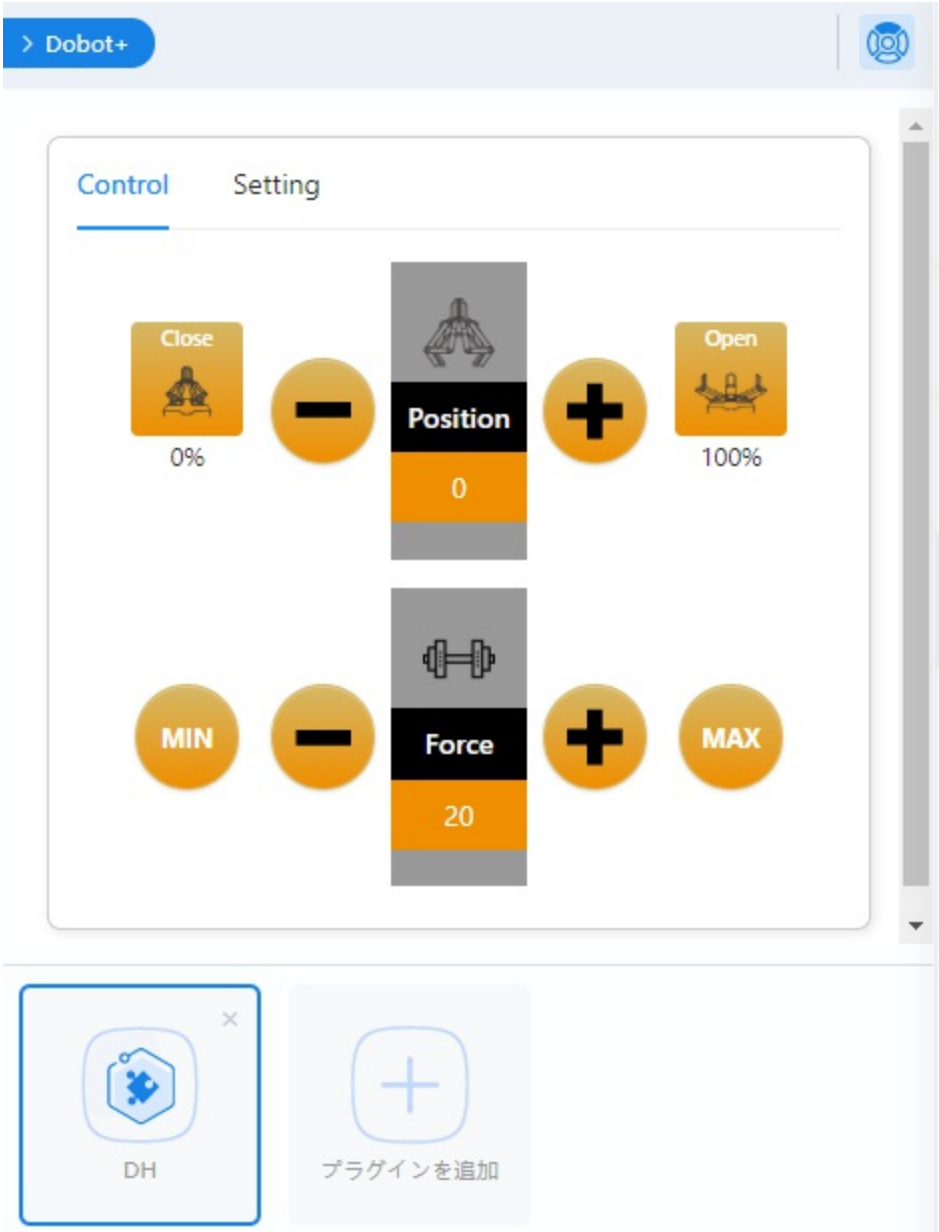
8 Dobot+

この画面は、ロボットアームの先端プラグインをインストールおよび構成するためのものであり、プラグインは先端ツールを制御することに使用できます。

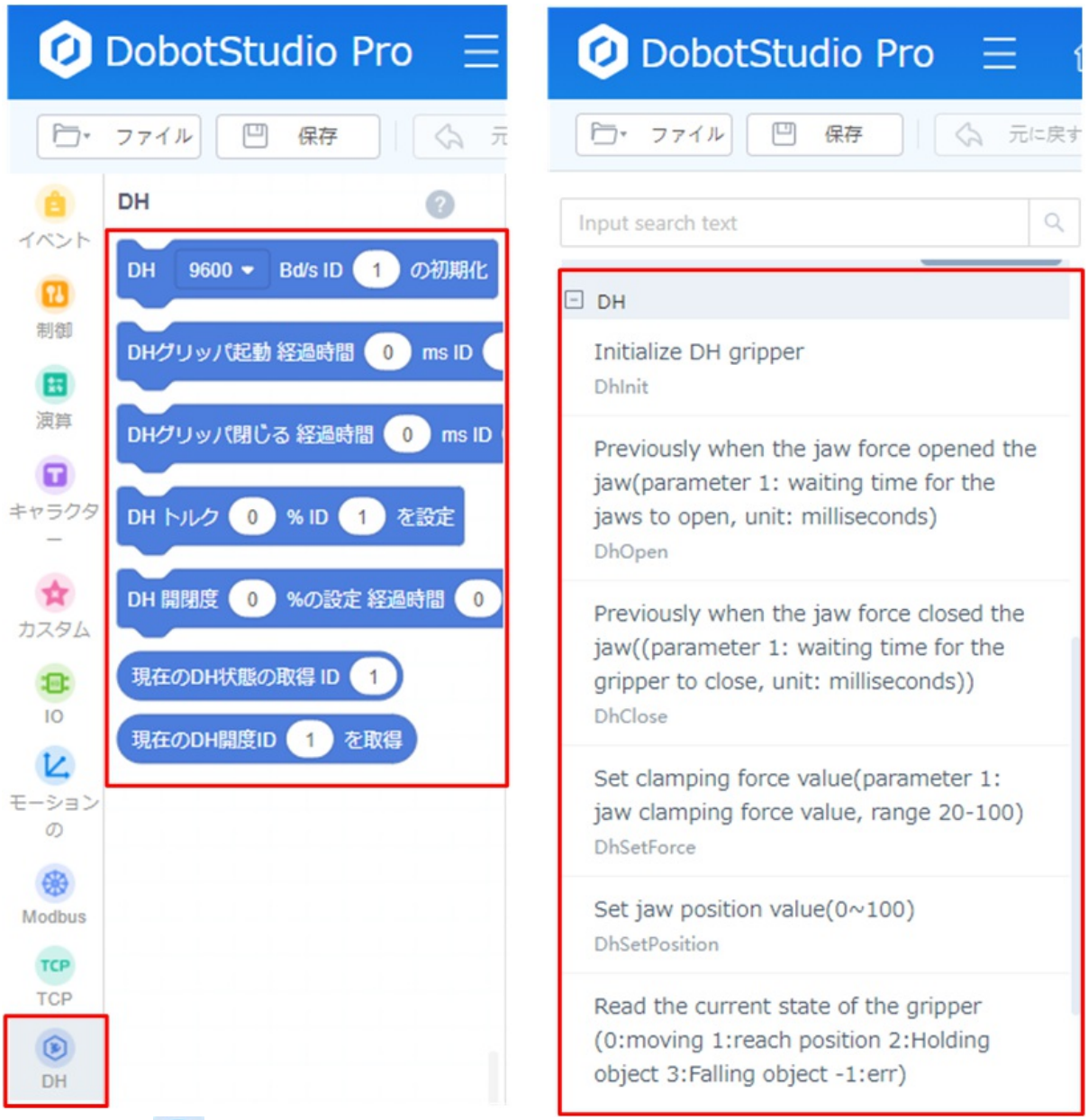
パネル左上の「Dobot+」をクリックすると、パネルを非表示にすることができます。非表示にした後、右側のツールバーの「Dobot+」ボタンをクリックすると、表示が戻ります。

「プラグインを追加」をクリックすると、プラグインリストを確認できます。プラグイン右側の「インストール」をクリックすると、そのプラグインをインストールできます。また、「プラグインをインポート」をクリックしてカスタムのプラグインをアップロードすることもできます。

プラグインのインストール完了後に、Dobot+のメイン画面に表示されます。対応するプラグインを選択した後、プラグインの基本機能を構成できます。それぞれのプラグインの構成方法は異なります。本文では詳細に説明しません。プラグインアイコンの右上にあるXをクリックすると、プラグインを削除できます。



プラグインを追加した後、グラフィカルプログラミングとスクリプトプログラミングにもプラグイン関連のブロック/インストラクションを新規追加できます。下図はDHグリッパのプラグインを例に挙げています。



パネル右上の  をクリックすると、プラグインにショートカットキーを設定できます。保存すると、ロボットアーム先端ボタンにより先端のツールを制御できます。

例えば、DHグリッパのプラグインを選択し、ショートカットーキー1にグリッパの開く操作を設定し、ショートカットキー2にグリッパの閉じる操作を設定し、保存します。その後、図示された先端ボタンを押して、グリッパを開き、再度先端ボタンを押すと閉じます。以後は循環します。

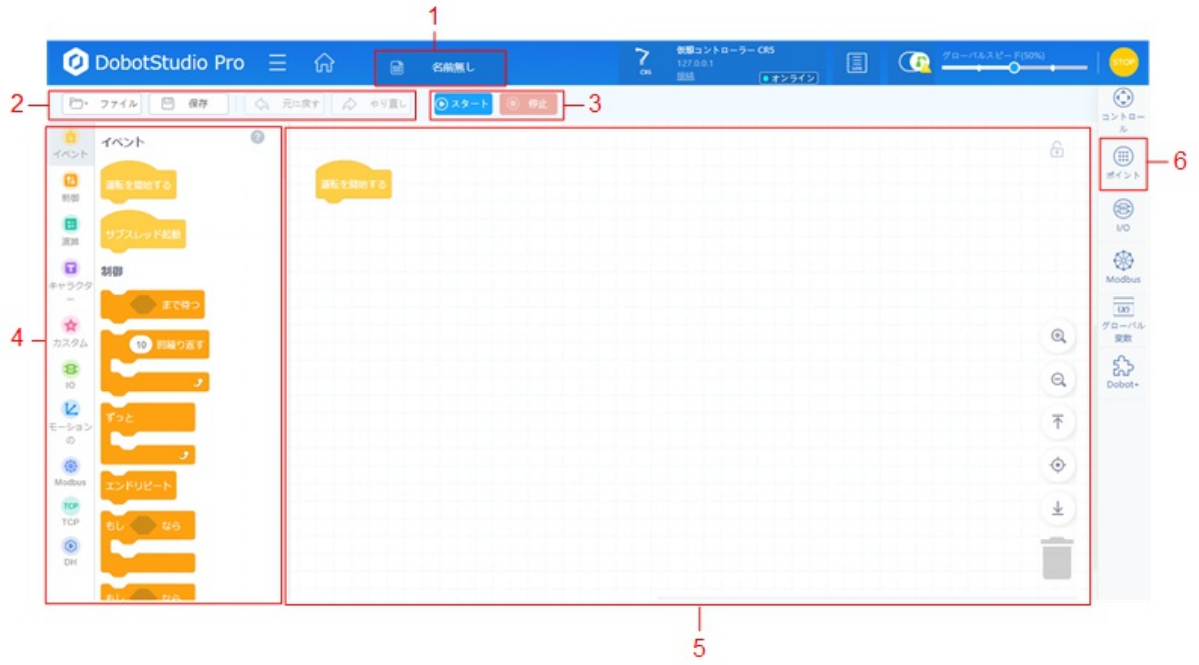

Dobot+
ショートカットキーを追加する
[終了ボタン
本機の終了ボタンを押すとショートカット1が実行され、もう一度押すとショートカット2が実行されます
プラグインリスト
DH
ショートカットキー1
DhOpen
ショートカットキー2
DhClose

9 プログラミング

9.1 グラフィカルプログラミング

DobotStudio Proはグラフィカルプログラミング能力を提供し、ユーザーはブロックをドラッグする方法によりプログラミングを行い、これによりロボットアームの動作を制御でき、直感的に分かりやすいです。

> 説明
>
> 本章では主にグラフィカルプログラミング画面の使用方法を紹介し、ブロックの具体的な説明については付録Bをご参照ください。



番号	説明
1	現在開いているプロジェクトの名称を表示します。
2	プロジェクトファイルの管理およびプログラミング操作に対する取り消しまたは復元に使用されます。 「ファイル」のプルダウンメニューでグラフィカルプログラミングのプロジェクトのスクリプトプログラミングのプロジェクトへの変換に対応します。変換後、スクリプトプログラミング画面で変換後のプロジェクトを開くことができます。
3	現在のプロジェクトのデバッグ、実行、一時停止と停止を制御するために使用されます。詳細はプロジェクトのデバッグと実行を参照してください。
4	プログラミングに必要なブロックを提供し、分類および色に応じて必要なブロックを検索することができます。 モジュール右上の「？」をクリックすると、ブロックの説明ドキュメントを確認できます。
	プログラムの作成エリアで、ブロックをこのエリアにドラッグすることによりプ

5	ログラムを作成できます。 このエリアに置いたブロックを右クリックするとメニューが開き、ブロックのコピー、削除およびイベントブロックを含まない1組のブロックのサブルーチンへの変換に対応します。 変更済みで未保存のブロック左側には  アイコンが現れ、ユーザーにこのブロックが変更済みであることを注意喚起します。
6	クリックすると点保存画面が開け、保存されていない変更がある場合、アイコン右下に赤い点が表示されます。詳細は点保存画面をご確認ください。

プログラミングエリア右側のアイコンの説明は以下のとおりです。

Icon	Description
	編集ステータスに入るために使用されます。 編集ステータスの下、ユーザーはブロックを複数またはすべて選択してコピーまたは削除できます。 「編集を取り消す」をクリックするか、プログラミングエリアでその他の操作をすると、編集ステータスを終了します。 
	プログラミングエリアをロック/ロック解除をするために使用されます。
	それぞれプログラミングエリアを拡大/縮小するために使用されます。
	それぞれブロックの上部に戻る/中央でブロックを表示する/ブロックの下部に戻るために使用されます。
	プログラミングエリア中のブロックをここにドラッグして削除します。 ユーザーはブロックを右クリックして、削除を選択することもできます。

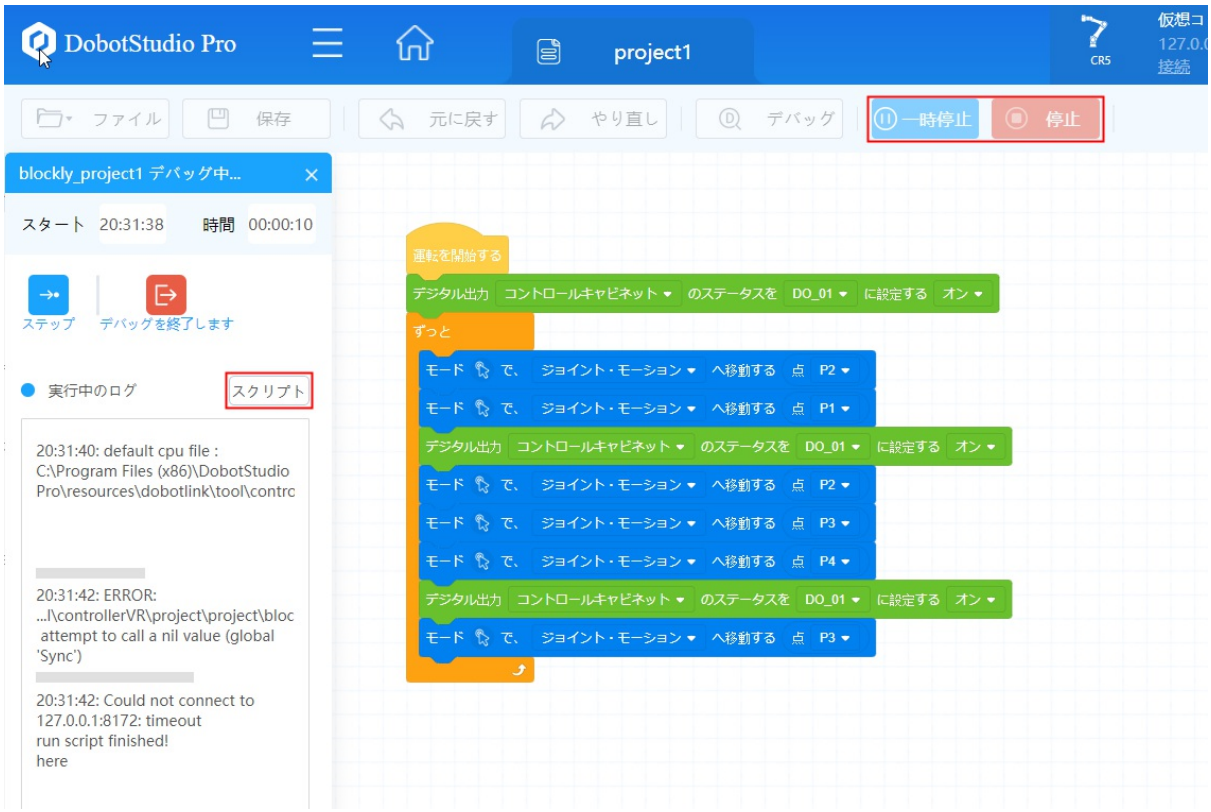
点保存画面

点保存画面は、下記に示すとおり、プログラミングの過程において使用するロボットアームの点位置を管理するために使用されます。

番号	説明
1	クリックすると点保存パネルを非表示にすることができます。非表示にした後、右側ツールバーの「点保存」ボタンをクリックすると再度表示されます。保存されていない変更がある場合、右上に赤色*が表示されます。
2	RunTo機能の移動モードを設定します。
3	ティーチングポイントリストファイルをインポートするか、現在のティーチングポイントリストをファイルとしてエクスポートします。
4	クリックすると、制御パネルを折りたたんで表示することができます。折りたたまれた状態で再度クリックすると、折りたたみをキャンセルできます。
5	点保存管理区域。 • ロボットアームを指定した点まで移動させた後、「新規追加」をクリックすると、ロボットアームの現在の点位置を新しいティーチングポイントとして保存できます。 • 1つのティーチングポイントを選択した後、このティーチングポイント「Name」以外のいずれかの値をダブルクリックすると、この値を直接変更できます。 • 1つのティーチングポイントを選択した後、「上書き」をクリックすると、ロボットアームの現在の点位置でこの保存点を上書きできます。 • ティーチングポイントを選択後、「RunTo」を長押しすると、ロボットアームをこの点まで移動させられます。 • 1つのティーチングポイントを選択後、「削除」をクリックすると、このティーチングポイントを削除できます。

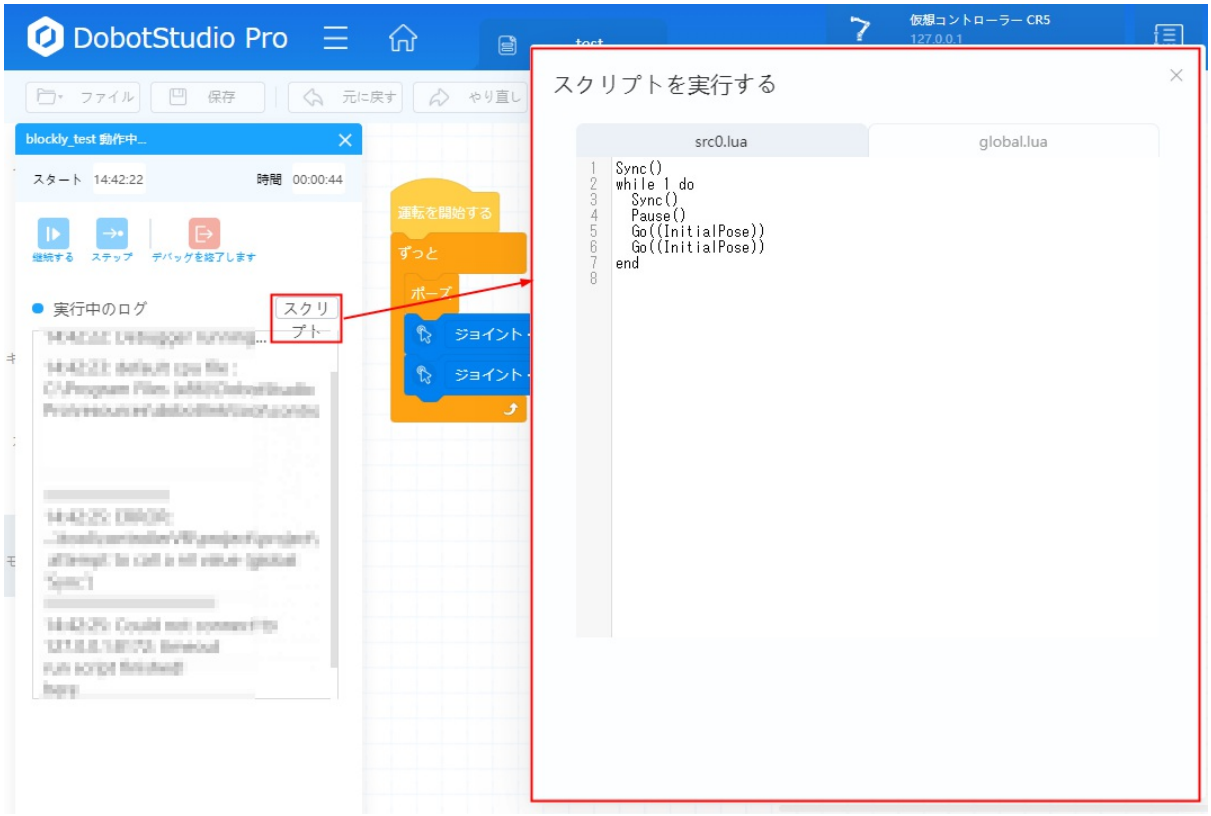
プロジェクトのデバッグと実行

プロジェクトを保存後、「デバッグ」をクリックすると、プロジェクトをシングルステップで実行し、実行過程におけるログが表示できます。 -「シングルステップ」をクリックすると、プロジェクトをシングルステップで実行でき、1回のクリックで1つのインストラクションを実行します。 -「デバッグを終了」をクリックすると、デバッグモードを終了できます。 -「スクリプト」をクリックすると、このプロジェクトに対応する実行スクリプトが表示できます。



プロジェクトを保存後、「開始」をクリックすると、プロジェクトの実行を開始し、実行過程におけるログが表示されます。

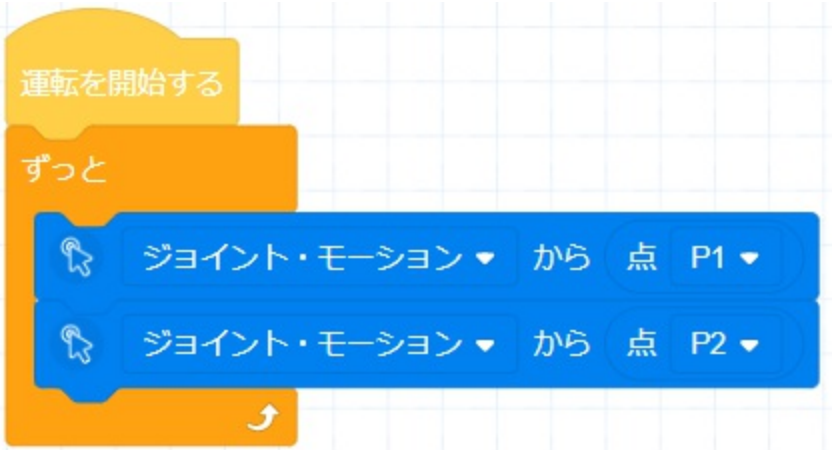
- 「一時停止」をクリックすると、プロジェクトの実行を一時停止でき、かつこのボタンは「続行」に変わります。クリックするとプロジェクトの実行を継続します。
- 「停止」をクリックすると、プロジェクトの実行を停止します。
- 「スクリプト」をクリックすると、このプロジェクトに対応する実行スクリプトが表示できます。



操作手順

以下では、ロボットアームの2つの点の間における循環運動を制御するプログラミング作成を例として、グラフィカルプログラミングの主な機能の使用方法を紹介します。

1. 点保存画面を開き、ロボットアームを任意の点P1まで移動させた後、「新規追加」をクリックして点P1の位置情報を保存します。
2. ロボットアームを別の任意の点P2に移動させた後、「新規追加」をクリックして点P2の位置情報を保存します。
3. 左側のブロックエリアから「繰り返し実行」ブロックを「実行開始」ブロックの下にドラッグします。
4. ポイント移動ブロックを「繰り返し実行」ブロックにドラッグし、目標点はP1を選択します。
5. さらに1つのポイント移動ブロックを前のポイント移動ブロックの下にドラッグし、目標点はP2を選択します。



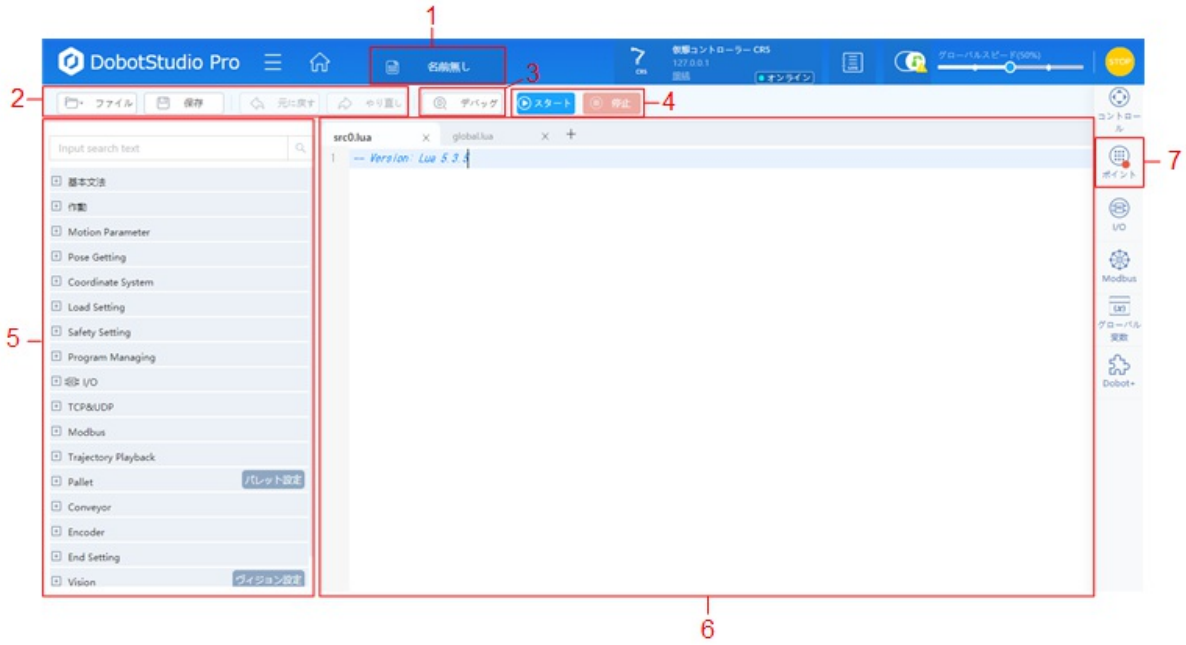
6. 「保存」をクリックして、プロジェクトの名称をカスタマイズし、「確認」をクリックします。

7. 「開始」をクリックすると、ロボットアームは移動を始めます。

9.2 スクリプトプログラミング

Dobotロボットアームは、例えば、移動インストラクション、TCP/UDPインストラクションなど、多数のDobot APIインターフェスを提供します。Lua言語を採用するため、ユーザーは二次開発中に容易に呼び出すことができます。DobotStudio ProはLuaスクリプトのプログラミング環境を提供し、ユーザーは自らLuaスクリプトを作成してロボットの動作を制御することができます。

> 説明
>
> 本章は主にスクリプトプログラミング画面の使用方法を説明し、具体的な関数説明については付属書Cを参照してください。



番号	説明
1	現在開いているプロジェクトの名称を表示します。
2	プロジェクトファイルの管理およびプログラミング操作に対する取り消しまたは復元に使用されます。
3	デバッグ画面を開くために使用されます。詳細については、デバッグプロジェクトを参照してください。
4	現在プロジェクトの実行および停止に使用されます。詳細については、プロジェクトの実行を参照してください。
	プログラミング関数の検索および使用に使用されます。 • 関数の左側の「?」をクリックすると、関数説明ドキュメントを表示することができます。

5	• 関数をダブルクリックすると、右側のプログラミングエリアにLuaインストラクションを追加することができます。 • 関数の右側に青色のボタンがある場合、青色のボタンをダブルクリックすると、右側のプログラミングエリアに詳細パラメータ付きのLuaインストラクションを追加することができます。 
6	プログラムの作成エリア。 • 「src0.lua」ファイルはメインスレッドであり、あらゆるインストラクションを呼び出すことができます。 • 「global.lua」ファイルは変数およびサブ関数の定義にのみ使用されます。 • 「+」をクリックすると、サブスレッドを追加することができます。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す。I/O、変数などを設定することができますが、移動インストラクションを呼び出すことはできません。
7	クリックすると点保存画面が開け、保存されていない変更がある場合、アイコン右下に赤い点が表示されます。詳細は点保存画面をご確認ください。

点保存画面

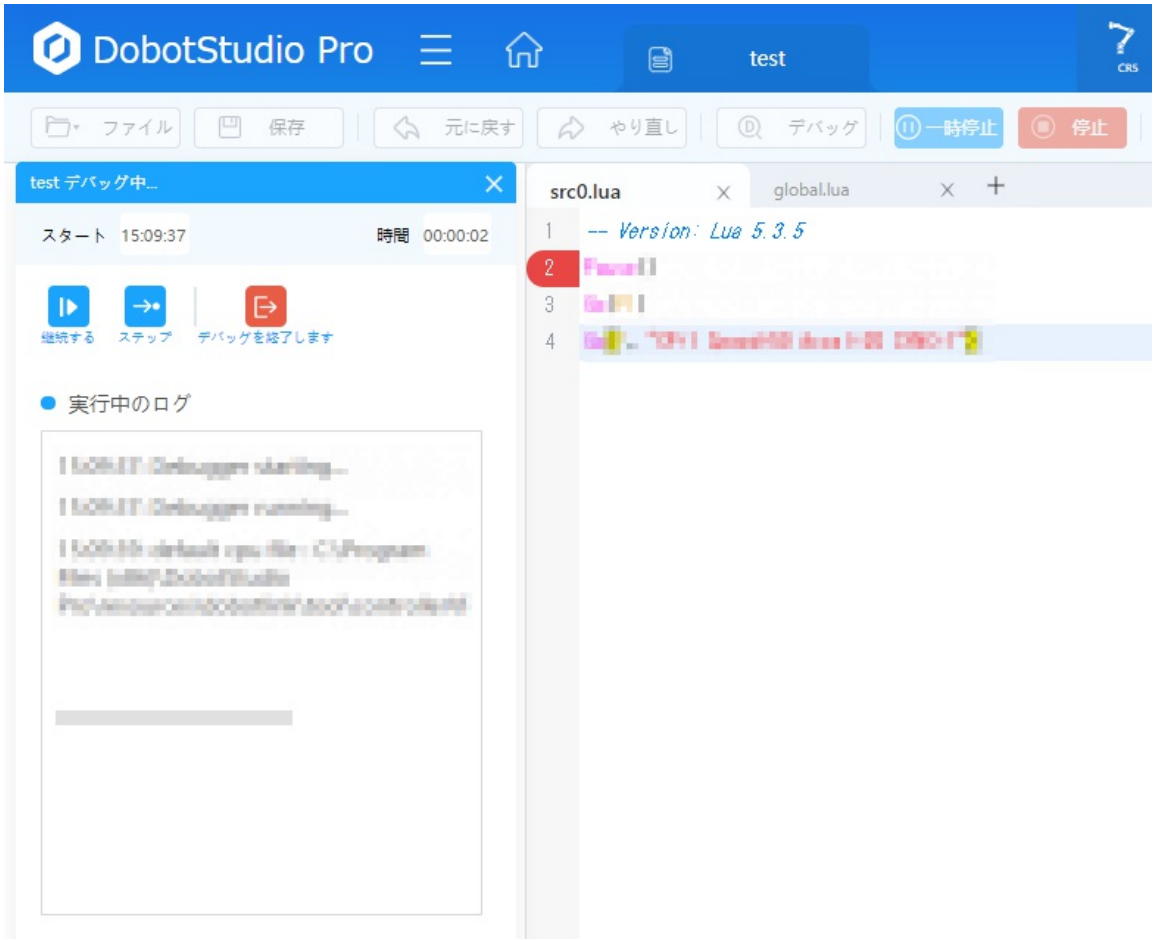
点保存画面は、下記に示すとおり、プログラミングの過程において使用するロボットアームの点位置を管理するために使用されます。

番号	説明
1	クリックすると点保存パネルを非表示にすることができます。非表示にした後、右側ツールバーの「点保存」ボタンをクリックすると再度表示されます。保存されていない変更がある場合、右上に赤色*が表示されます。
2	RunTo機能の移動モードを設定します。
3	ティーチングポイントリストファイルをインポートするか、現在のティーチングポイントリストをファイルとしてエクスポートします。
4	クリックすると、制御パネルを折りたたんで表示することができます。折りたたまれた状態で再度クリックすると、折りたたみをキャンセルできます。
5	点保存管理区域。 • ロボットアームを指定した点まで移動させた後、「新規追加」をクリックすると、ロボットアームの現在の点位置を新しいティーチングポイントとして保存できます。 • 1つのティーチングポイントを選択した後、「上書き」をクリックすると、ロボットアームの現在の点位置でこの保存点を上書きできます。 • ティーチングポイントを選択後、「RunTo」を長押しすると、ロボットアームをこの点まで移動させられます。 • 1つのティーチングポイントを選択後、「削除」をクリックすると、このティーチングポイントを削除できます。

デバッグプロジェクト

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

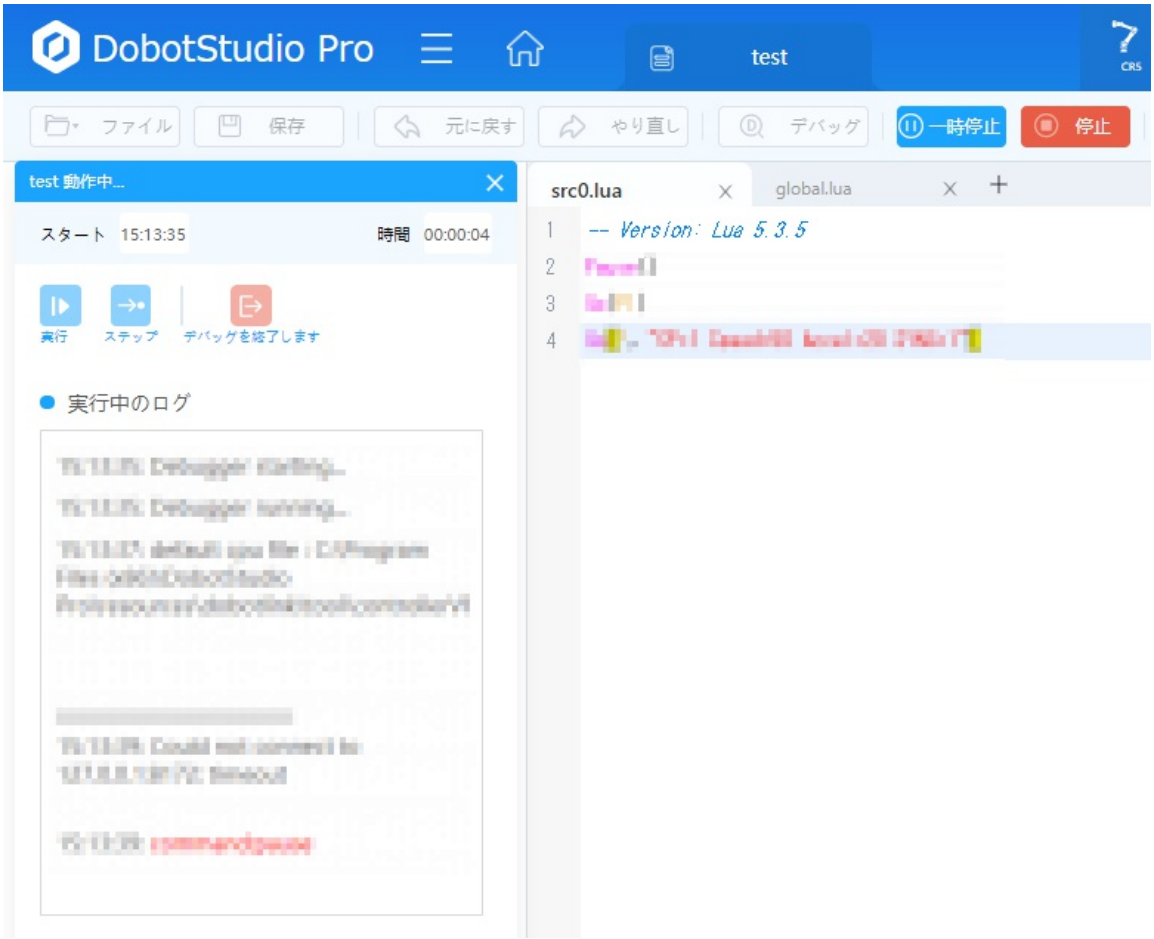
- コードの左側のライン番号をクリックするとブレークポイントを設定することができます。プログラムはデバッグモードでブレークポイントに実行すると、自動的に一時停止します。
- プログラムがブレークポイントで一時停止した後に、「続行」をクリックすると、プログラムは次の実行に進みます。「単一ステップ」をクリックすると、プログラムは単一ステップずつ実行し、クリックするごとに1つのインストラクションを実行します。
- 「デバッグを終了」をクリックすると、デバッグモードを終了できます。



プロジェクトの実行

プロジェクトを保存後、「開始」をクリックすると、プロジェクトの実行を開始し、実行過程におけるログが表示されます。

- 「一時停止」をクリックすると、プロジェクトの実行を一時停止でき、かつこのボタンは「続行」に変わります。クリックするとプロジェクトの実行を継続します。
- 「停止」をクリックすると、プロジェクトの実行を停止します。



操作手順

次の文書で、2つの点の間でのループ移動をするようにロボットアームを制御するプログラムの作成を例として、スクリプトプログラミングの主要機能の使用方法を説明します。

1. 点保存画面を開き、ロボットアームを任意の点P1まで移動させた後、「新規追加」をクリックして点P1の位置情報を保存します。
2. ロボットアームを別の任意の点P2に移動させた後、「新規追加」をクリックして点P2の位置情報を保存します。
3. プログラミングエリアにループインストラクションを追加します。
4. ループコードの「end」の前に動きインストラクションを追加し、目標点はP1とします。
5. さらにもう一個の動きインストラクションを追加し、目標点としてP2を選択します。

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

6. 「保存」をクリックして、プロジェクトの名称をカスタマイズし、「確認」をクリックします。
7. 「開始」をクリックすると、ロボットアームは移動を始めます。

10 プロセス

10.1 軌跡再現

軌跡再現は、ユーザーがロボットアームをドラッグしてティーチングした軌跡再現に使用されます。そのメイン画面は下図に示すとおりです。

右上角の「軌跡の新規作成」をクリックし、ロボットアーム先端の指示ランプが黄色常時点灯に変わると、ロボットアーム本体をドラッグすることができます。ドラッグした軌跡は記録され、再現のために使用されます。

記録したい軌跡が完了したら、「軌跡の保存」をクリックし、ロボットアーム先端の指示ランプが緑色常時点灯に変わると、軌跡ファイル一覧に新しい記録が追加されます。

保存済み軌跡をクリックして選択すると、「削除」、「名前の変更」と「軌跡の再現」ボタンが現われます。

- 「軌跡再現」をクリックするとロボットアームに記録された軌跡を再現させることができます。このときロボットアーム先端の指示ランプは黄色点滅に変わり、自動移動を開始します。軌跡再現が完了すると、ロボットアームは移動を停止し、先端の指示ランプは緑色常時点灯に変わります。
- 「名前の変更」をクリックすると、軌跡ファイル名を変更することができます。
- 「削除」をクリックすると、該当軌跡を削除することができます。

> 説明
>
> - 保存済みの軌跡ファイルは、グラフィカルプログラミングおよびスクリプトプログラミング中に軌跡再現関連インストラクションで呼び出すことができます。
> - 軌跡の再現は安全IO信号の影響を受けません。

「詳細設定」をクリックすると、軌跡再現方法を設定することができます。「等速で再現しますか？」を選択しないときは、再現レートを設定することができます。

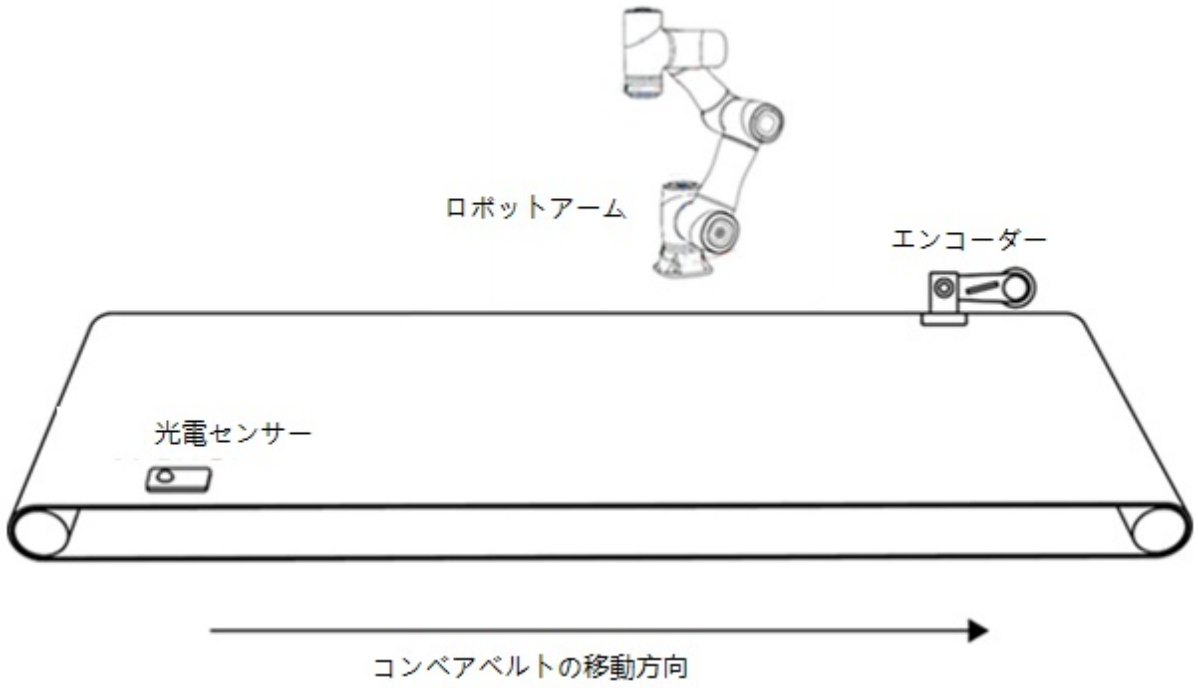

高度な設定
均一に再現されるかどうか
再発を追跡する
1
スピード倍率
1
キャンセル
はい

10.2 コンベアベルト

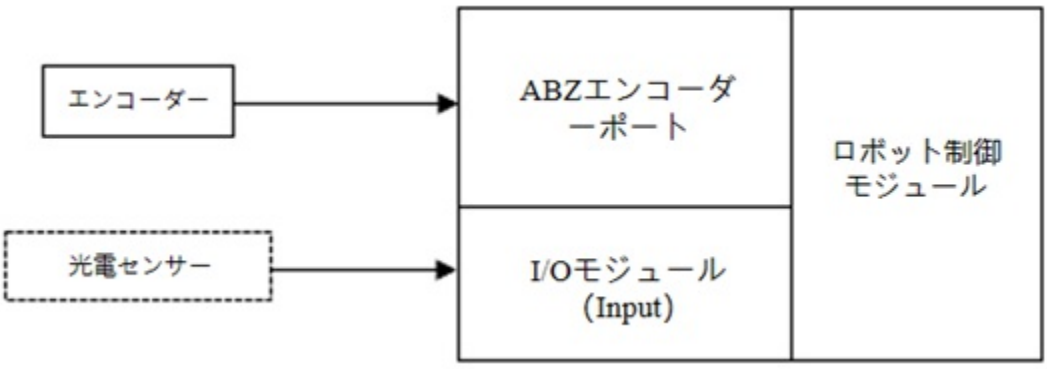
コンベアベルトプロセスは光電センサーを介してコンベアベルト上の物体を検出でき、ロボットアームは物体の移動過程においてそれを正確にピックしたり、その他の応用をしたりします。

1 環境の構築

完全なコンベアベルトプロセスの環境は、下図に示すとおりです。



通信見取り図は以下のとおりです。



エンコーダー

エンコーダーはコンベアベルトの移動距離とワークの位置を記録し、ロボット制御システムにフィードバックするために使用され、ロボットアームがピークを行うのに便利です。エンコーダーをコントロールキャビネットのABZエンコーダーインターフェスと接続します。E6B2-

CWZ1X（1000P/R）エンコーダーを選択することをお勧めします。ピンの接続は下表に示すとおりです。

色	ポート
黒	A+
黒/赤	A-
白	B+
白/赤	B-
橙	Z+
橙/赤	Z-
茶	I/O 5V
青	I/O 0V

光電センサー

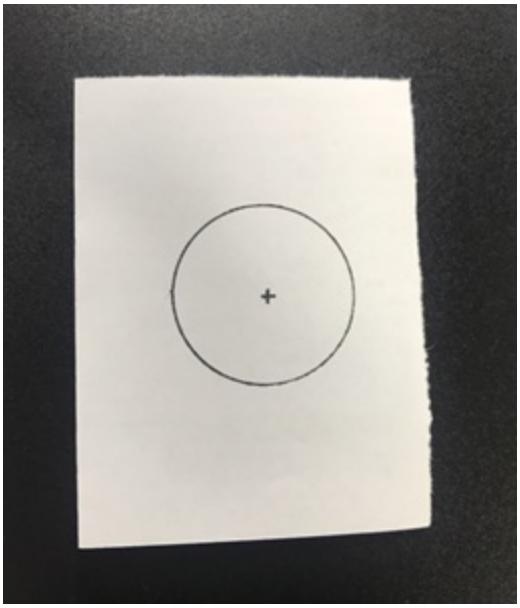
光電センサーが物体の有無を検出する時、さまざまなレベル信号を出力し、制御システムは信号エッジにより物体を検出できます。光電センサーをコントロールキャビネットのI/Oモジュールにおける任意のDIインターフェスに接続します。

2 コンベアベルトのキャリブレーション

コンベアベルトプロセスを構成する前に、座標系を設定し、コンベアベルトをキャリブレーションする必要があります。これはコンベアベルトとロボットアームの間の相対的関係を説明するために使用されます。下記のキャリブレーションの過程において、ユーザー座標系を採用してキャリブレーションを行います。

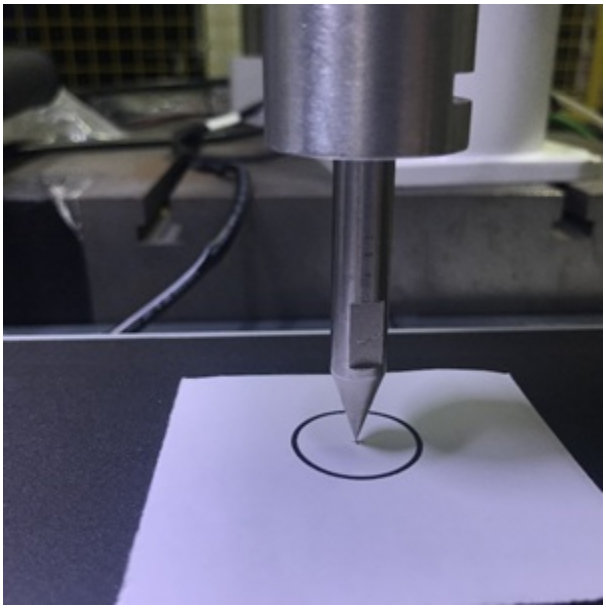
キャリブレーション前にロボットアームの先端にキャリブレーションピンを取り付け、キャリブレーションラベルを準備する必要があります。本ドキュメントでは詳細に説明しません。

1. 下図に示すとおりに、コンベアベルトにラベルを貼ります。



2. ユーザー座標系設定画面に進みます。

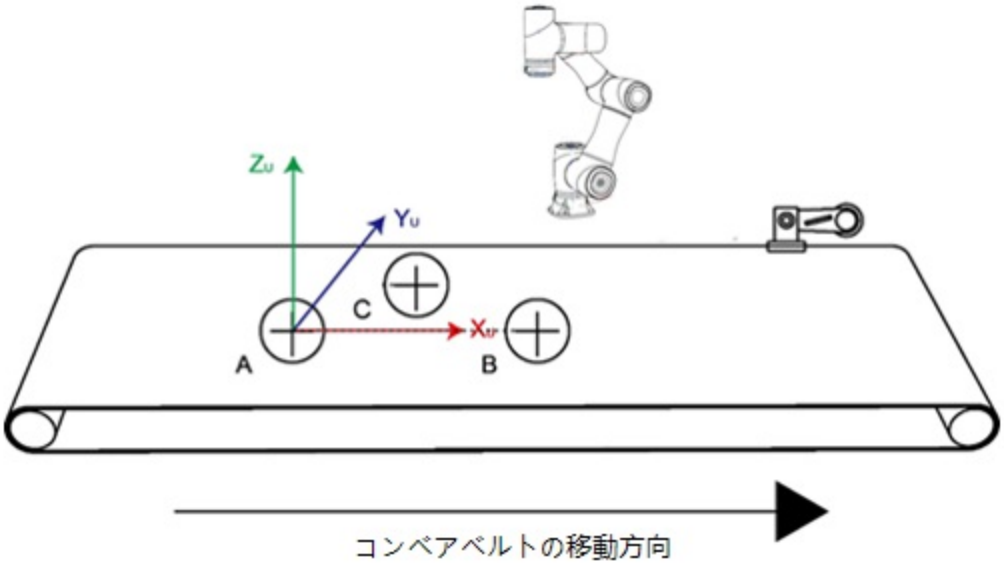
3. ロボットアームをイネーブルにし、コンベアベルトのラベル部分までジョグし、最初の点（点A、座標系の原点）を取得します。



4. コンベアベルトを制御し、一定の距離を移動させた後停止し、ラベルもそれに伴い移動します。

5. ロボットアームをラベル部分までジョグし、2番目の点（点B、X軸を確定するために使用する）を取得します。

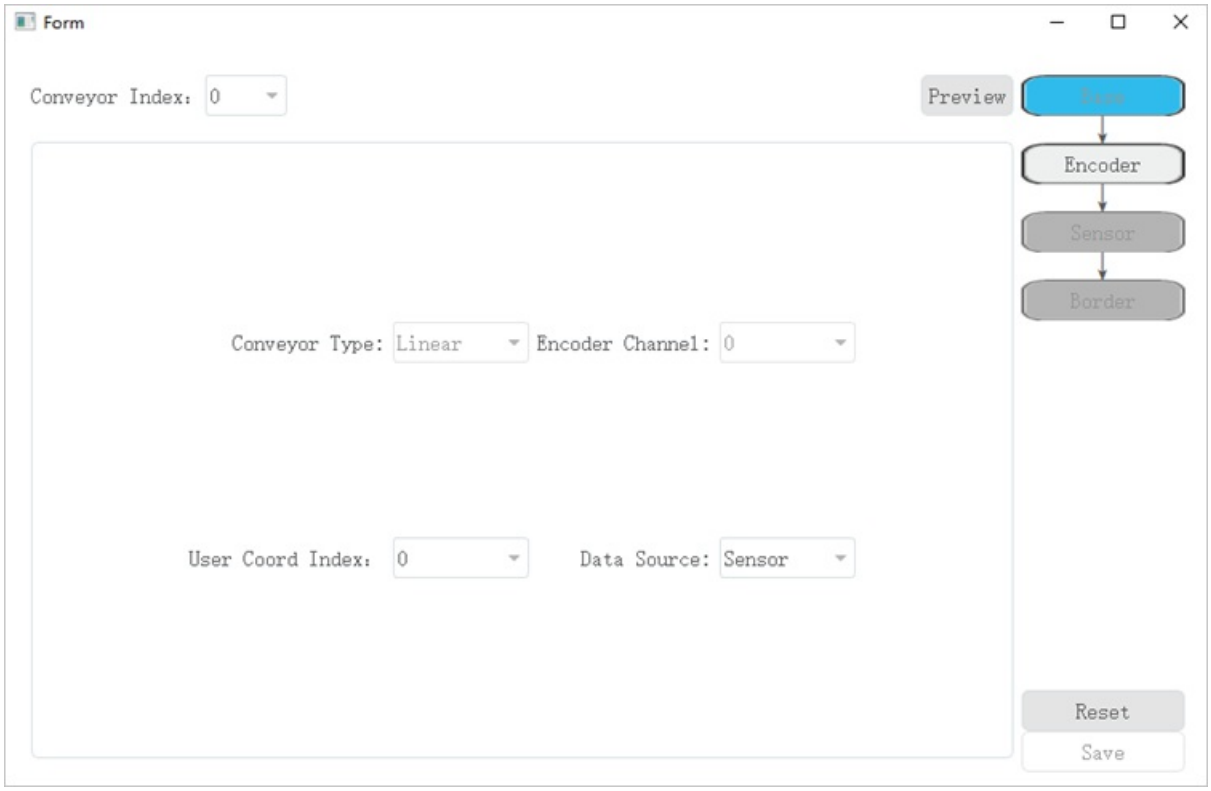
6. ロボットアームをコンベアベルト上の点ABにより確定された直線以外の任意の位置までジョグし、3つ目の点（点C、Y軸を確定するために使用する）を取得します。



7. 座標系を追加または上書きし、保存します。

3 コンベアベルトの構成

プロセス画面で「コンベアベルト追跡」を選択し、その後、「ツールを開く」をクリックすると、以下の画面が開きます。

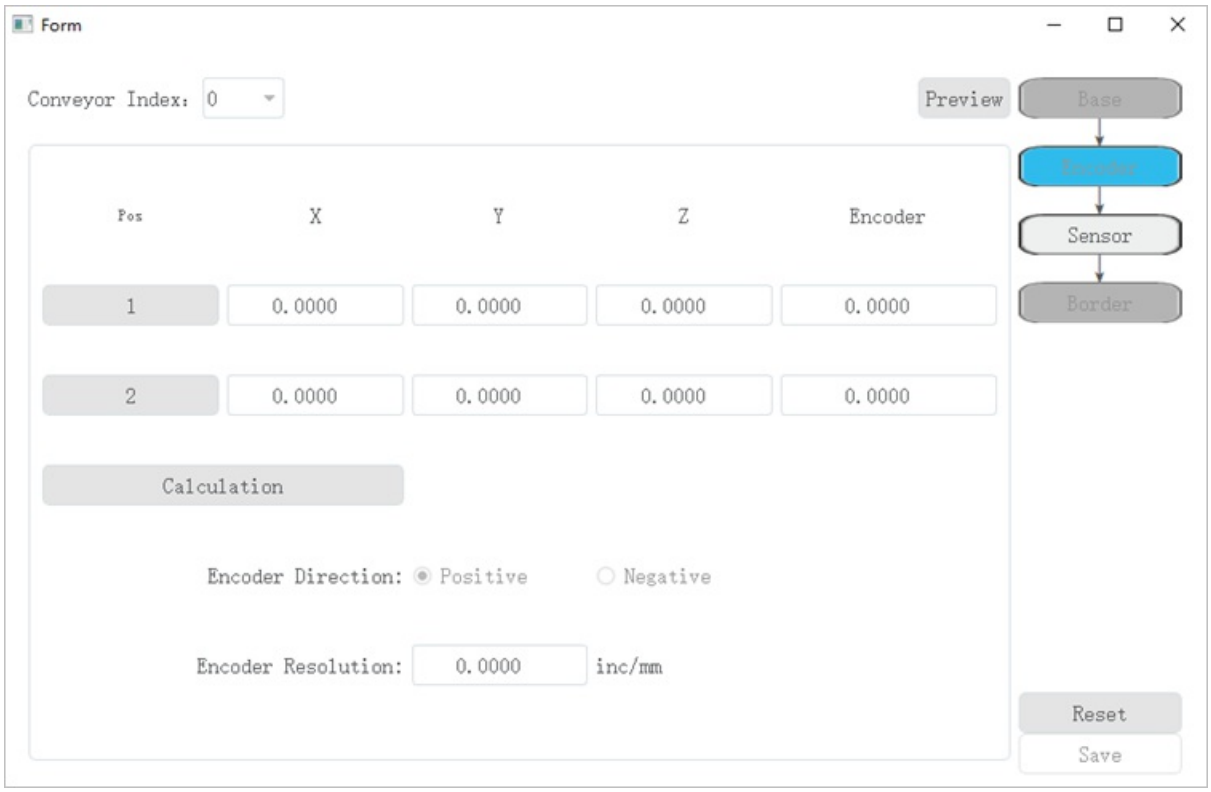


- コンベアベルト番号：構成するコンベアベルトの番号を選択します。複数のコンベアベルトの構成に対応しています。
- コンベアベルトのタイプ：現在、リニアコンベアベルトのみに対応し、構成できません。
- エンコーダーのチャンネル：現在、デフォルトのチャンネルにのみ対応し、構成できませ

ん。

- ユーザー座標系：前の手順で保存したコンベアベルトの座標系を選択します。
- 検出方式：現在、センサーによる検出にのみ対応し、構成できません。

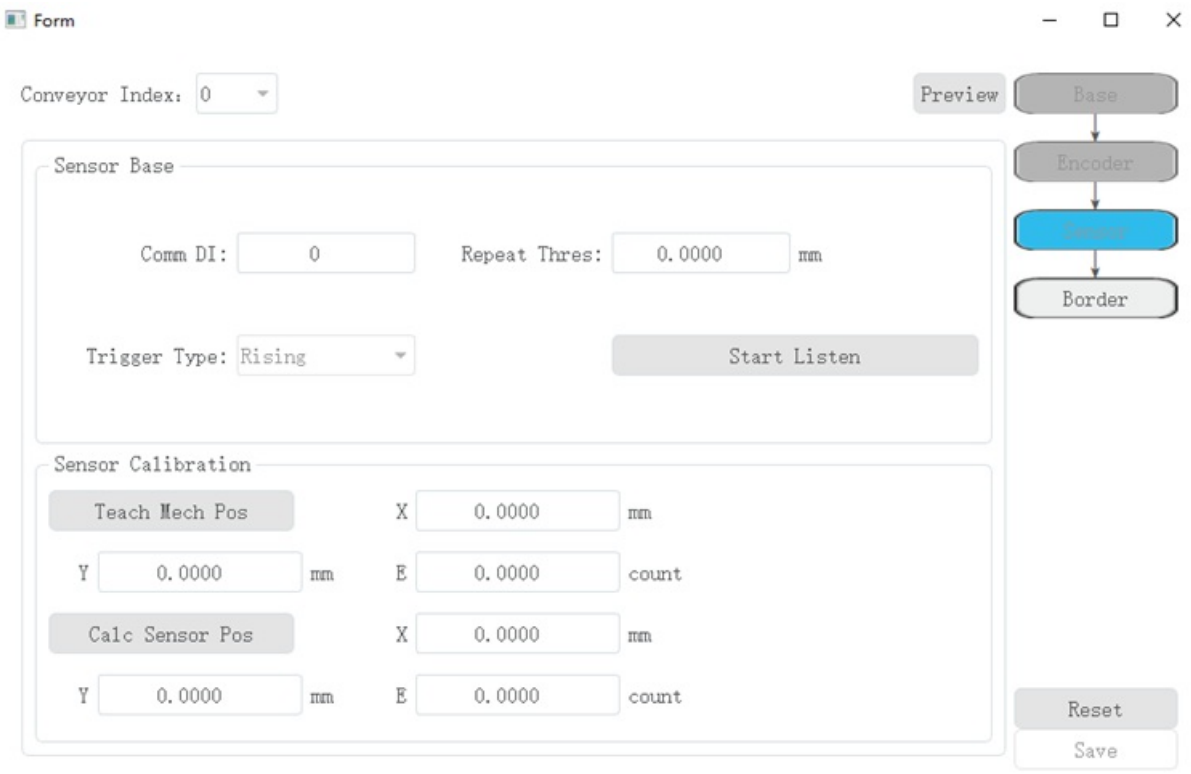
エンコーダーの設定



エンコーダー分解能のキャリブレーションエンコーダー分解能とは、コンベアベルトが単位長さ（mm）を移動する時のエンコーダーのパルス増分です。

1. コンベアベルトにラベルを貼り、ロボットアームをコンベアベルト上のラベル部分までジョグし、その後位置列の「1」をクリックして、位置1の値を取得します。
2. コンベアベルトを制御し、一定の距離を移動して停止し、ラベルもそれに伴い移動します。
3. ロボットアームをコンベアベルト上のラベル部分までジョグし、その後位置列の「2」をクリックして、位置2の値を取得します。
4. 実際の状況に応じてエンコーダーの方向を設定し、ここでは順方向を例とします。
5. 「計算」をクリックし、エンコーダーの分解能を導出します。

センサーの設定

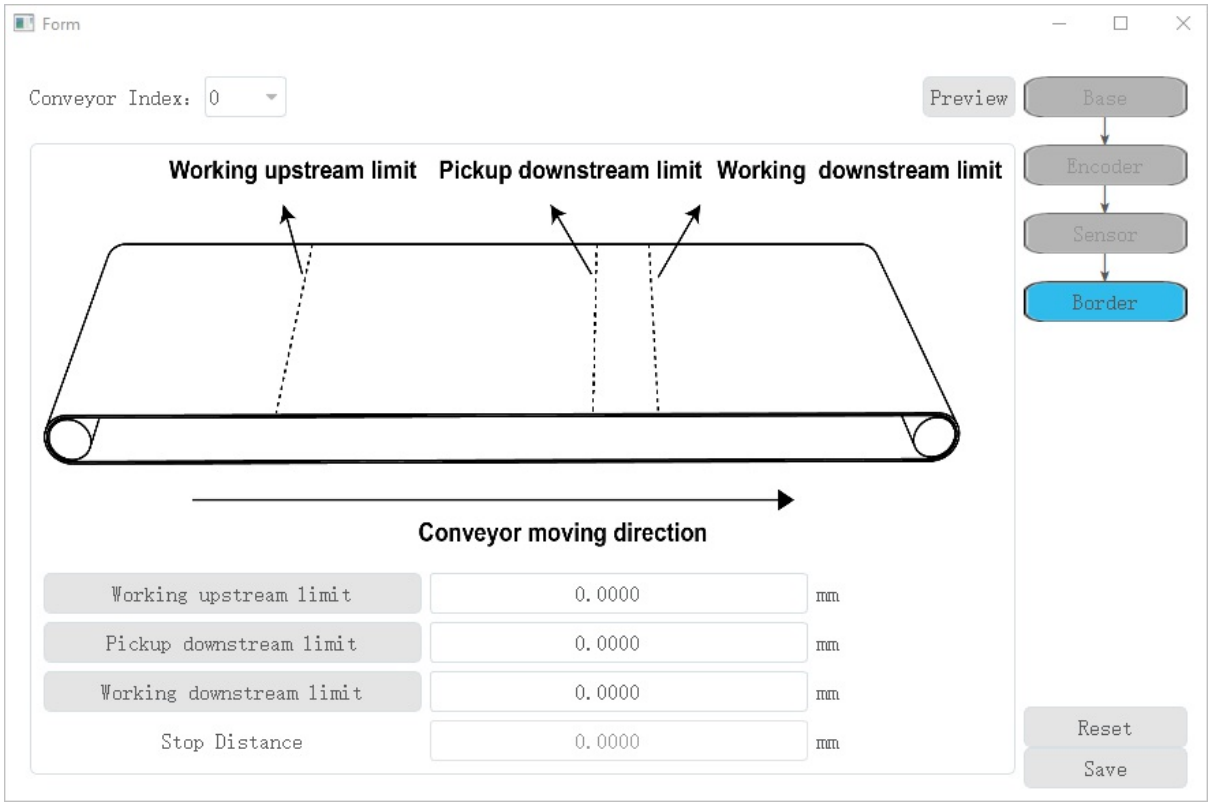


- 入力ポートのトリガー：センサーが接続するコントロールキャビネットのDIポートの番号を設定します。
- 重複排除距離：重複排除とは即ち1つの有効信号を検出した後、以降の一定の距離内でこの信号の変化を再度検出した場合、この信号は無効信号であると見なし、自動的に除去することです。実際の状況に応じて設定する必要があり、一般的には2mm～5mmに設定可能です。
- トリガーのタイプ：現在、信号の上昇エッジによるトリガーのみに対応しており、構成できません。

センサーのキャリブレーションはセンサーがワークによりトリガーされた瞬間、ワークの位置（即ちセンサーの位置）を取得するためで、それによりワークがコンベアベルトに従って移動する時、座標系のオフセットに基づき各時刻におけるユーザー座標系でのワークの位置を算出します。

1. 「リスニング開始」をクリックし、センサー信号のリスニングを開始します。
2. 1つのワークをセンサーの上流に配置し、コンベアベルトを起動すると、ワークがコンベアベルトと共に移動します。ワークがセンサーを経てロボットの動作範囲内まで移動した後、コンベアベルトを停止します。
3. ロボットアームをワークの中心位置までジョグし、「ティーチングポイント位置」をクリックすると、ワークの現在の位置を読み取ります。
4. 「センサー位置を計算」をクリックし、センサーの位置を導出します。

動作境界の設定



動作境界はロボットアームがコンベアベルトにおいて動作可能な範囲を設定するために使用されます。

- ワークが境界に入る：ワークがこの境界を越えた後、ロボットアームはワークを追跡し、ピックアップします。
- ピックアップの下限境界：ワークがこの境界を越えても、ロボットアームがこのワークのピックアップを開始していない場合、このワークの処理フローを完成できなかったものと判断し、このワークのピックアップを試行しません。この範囲はコンベアベルトの移動速度および実際の経験に基づき設定する必要があります。何回も試運転して最適値を取得することをお勧めします。
- ワークが境界を離れる：ワークがこの境界を越えた後、ロボットアームはワークの追跡を停止し（ロボットの運動区域外まで追跡するのを防止するため）、アラームをトリガーします。

各境界の対応する位置までそれぞれロボットアームをジョグし、ボタンをクリックして設定を行います。

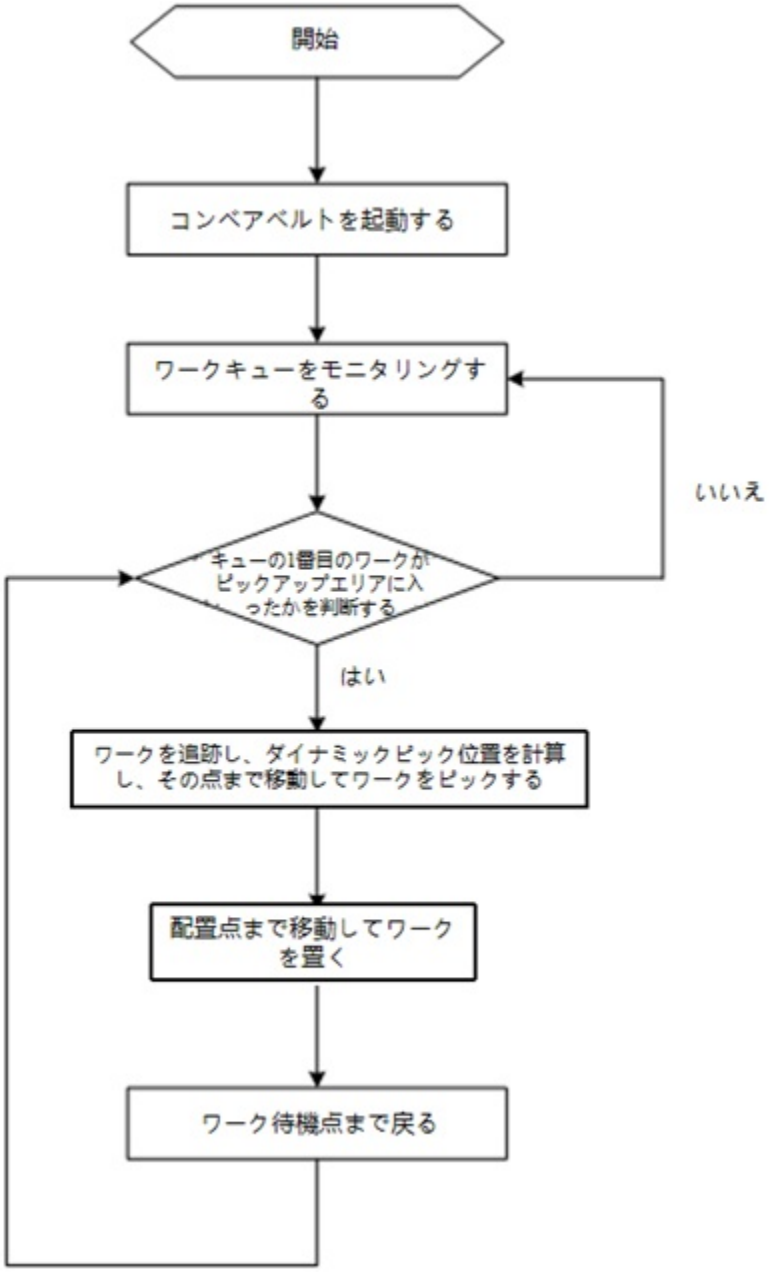
設定が完了後、「保存」をクリックすると、その後、プロジェクトでこのコンベアベルトの構成を使用できます。

「リセット」をクリックすると、すべての設定済みの内容がクリアできます。

4 応用事例

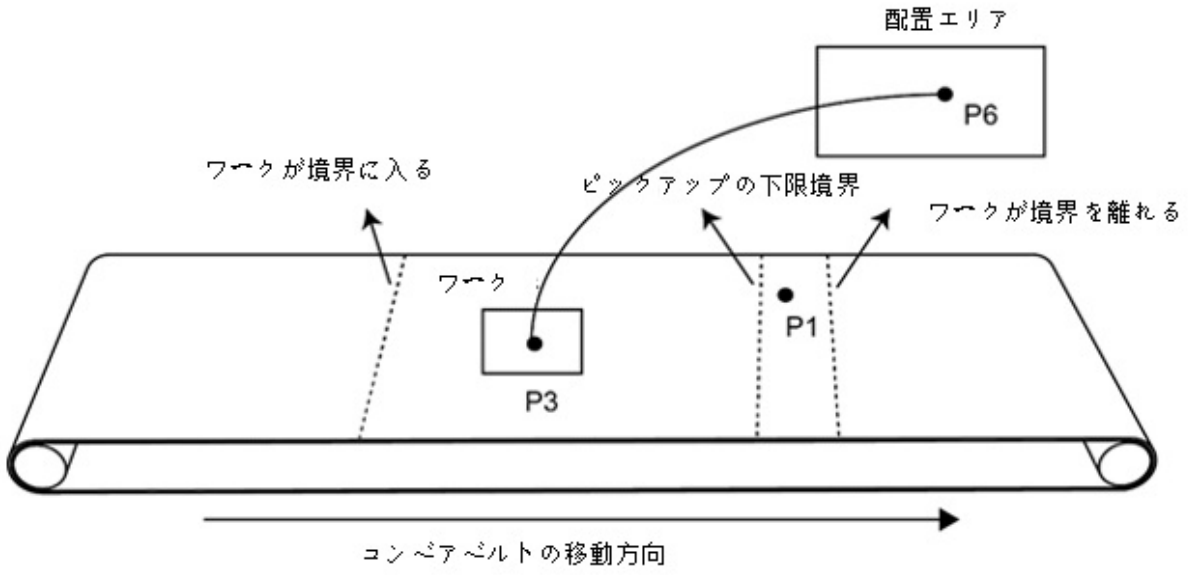
コンベアベルトのパラメータの構成を完了した後、コンベアベルト追跡APIを呼び出してスクリプトを作成することによりコンベアベルトの追跡作業が実現できます。

コンベアベルト追跡作業の作業フローは以下のとおりです。



本事例のプロジェクトスクリプトを作成する時、コンベアベルトを構成する時に設定したユーザー座標系の下で6つの点をティーチングする必要があります。

- ワーク待機点：P1
- ワーク追跡上方点：P2
- ワークピーク点：P3
- ワーク持ち上げ点：P4
- 配置上方点：P5
- 配置点：P6



事例スクリプトは以下のとおりです。

```
CnvVison(0) ----------------------//コンベアベルトを起動する

DO(9,0) ----------------------//DO1とDO2によりエアポンプの起動・停止と状態を制御する

DO(2,0)

local flag  ----------------------//ワークの標識が存在するか

local typeObject  ----------------------//ワークの属性分類

local point = {0,0,0}  ----------------------//追跡キューのワークの座標系

while true do

      Go(P1,"Speed=100 Accel=100 SYNC=1") ---//ロボットアームの待機点で、一般的には動作スペース内に設定する

      print("Test")

      while true do

                flag,typeObject,point = GetCnvObject(0,0) --//ワークの有無を照会し、ある場合はサイクル外にする

                if flag == true then

                          break

                end

                Sleep(20)

      end
```

```
        SyncCnv(0) ----------------------//コンベアベルトを同期し、追跡を開始する
        Move(P2,"SpeedS=100 AccelS=100") ------------------//ワーク上方点を追跡する
        Move(P3,"SpeedS=100 AccelS=100") ------------------//ワークピーク点
        Wait(100)
        DO(9,1) ----------------------//エアポンプを起動し、ワークをピークする
        --DO(2,1)
        Wait(100)
        Move(P4,"SpeedS=100 AccelS=100 SYNC=1")  ---------//ワーク持ち上げ点
        StopSyncCnv() ----------------------//コンベアベルト追跡を停止する
        Sleep(20)
        Go(P5,"Speed=100 Accel=100")  ----------------//配置点上方
        Go(P6,"Speed=100 Accel=100 SYNC=1")  --------//配置点
        Sleep(1000)
        DO(1,0) ----------------------//エアポンプをオフにする
        DO(2,0,"SYNC=1")
        Sleep(1000)
        Go(P5,"Speed=100 Accel=100")
end
```

11 ベストプラクティス

本章では、DobotStudio Proの各機能が連携して使用される方法を理解する手助けのため、リモートI/Oによりロボットアームを制御する完全なフローについて説明します。

フローを理解しやすくするため、まず1つのサンプルシナリオをプリセットしましょう。開始ボタンを押すと、動作指示ランプが点灯し、その後、ロボットアームが先端のグリッパを介して物品ピック点から資材をつかみ取り、積み下ろし点まで移動して資材を下ろします。その後、また物品ピック点に戻って資材をつかみ取り、このサイクルを繰り返します。

上記のシナリオを実現するため、ロボットアームの先端にグリッパを取り付け（DHシリーズを例とする場合、取付方法はDHシリーズのグリッパ使用マニュアルを参照してください）、ボタンと表示灯をコントロールキャビネットのI/Oポートに接続します（開始ボタンがDI11に、停止ボタンがDI12に、動作指示ランプがDO11に、アラーム指示ランプがDO12に接続されるとします。接続方法はロボットに対応するハードウェアユーザーマニュアルを参照してください）。

フロー全体

ハードウェアのインストールが完了し、かつロボットアームの電源が入れた後、ソフトウェア操作のフロー全体は以下のとおりです。

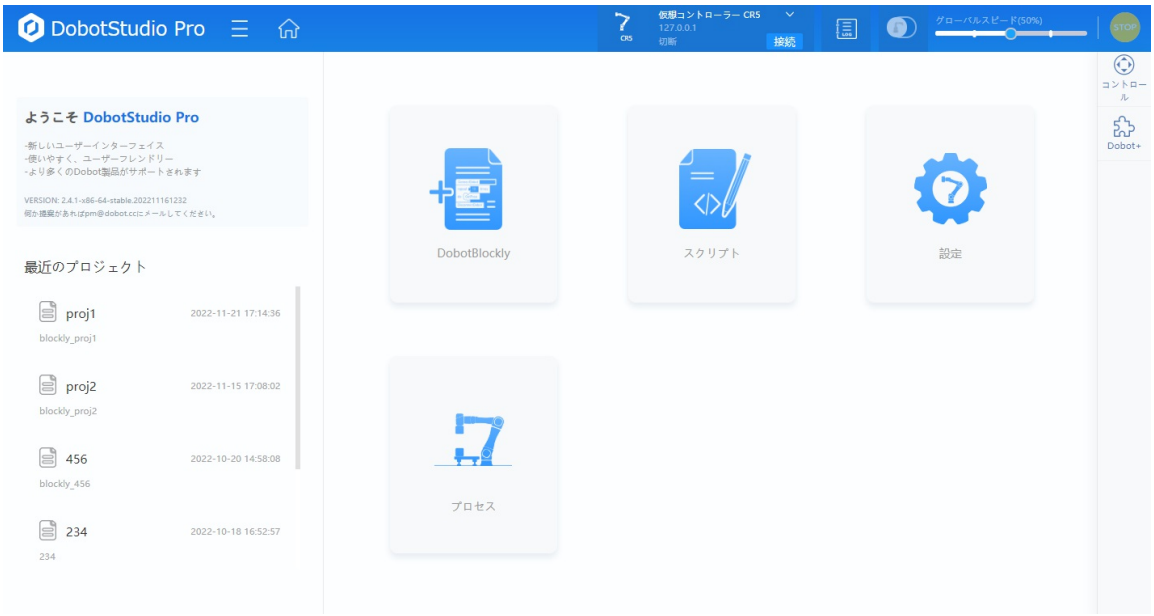
1. ロボットを接続する
2. Dobot+プラグインをインストールする（オプション）
3. ツール座標系を設定および選択する
4. プロジェクトを作成する
5. リモートI/Oモードを構成し、ログインする

操作手順

ロボットを接続し、イネーブルにする

ロボットアームの接続に関する詳細な説明は、ロボットの接続の章節を参照してください。ここでは無線接続を例に簡単に説明します。

1. PC端末でロボットのWi-Fiを検索し接続します。Wi-FiのデフォルトSSIDは「Dobot_WIFI_XXX」で、そのうち、XXXはロボットアームの台座に位置するロボットアーム番号です。初期のWi-Fiのパスワードは「1234567890」です。
2. DobotStudio Pro画面の上部で接続するロボットを選択し、その後、「接続」をクリックしてください。



3. イネーブルボタンをクリックし、負荷パラメータを設定した後、ロボットアームをイネーブルにします。

Dobot+プラグインをインストールする（オプション）

Dobot+プラグインに関する詳細な説明はDobot+の章節を参照してください。ここではDHプラグインのインストールを例に簡単に説明します。

インストールされた先端ツールに対応するプラグインがない場合は、この手順をスキップして、以降のプログラミング時に直接I/Oインストラクションにより先端ツールを制御します。

1. 画面右側のDobot+ボタンをクリックして、Dobot+パネルを開きます。
2. 「プラグインを追加」をクリックした後、DHプラグインの「インストール」ボタンをクリックすれば、インストールが完了します。

ツール座標系を設定および選択する

ツール座標系の詳細な説明はツール座標系の章節を参照してください。ここでは入力設定を例に簡単に説明します。

1. 「設定＞座標系管理＞ツール座標系」ページを開きます。
2. 座標系を新規追加または変更し、フランジ中心点に相対するツール中心点のオフセット量を入力し、確定をクリックします。

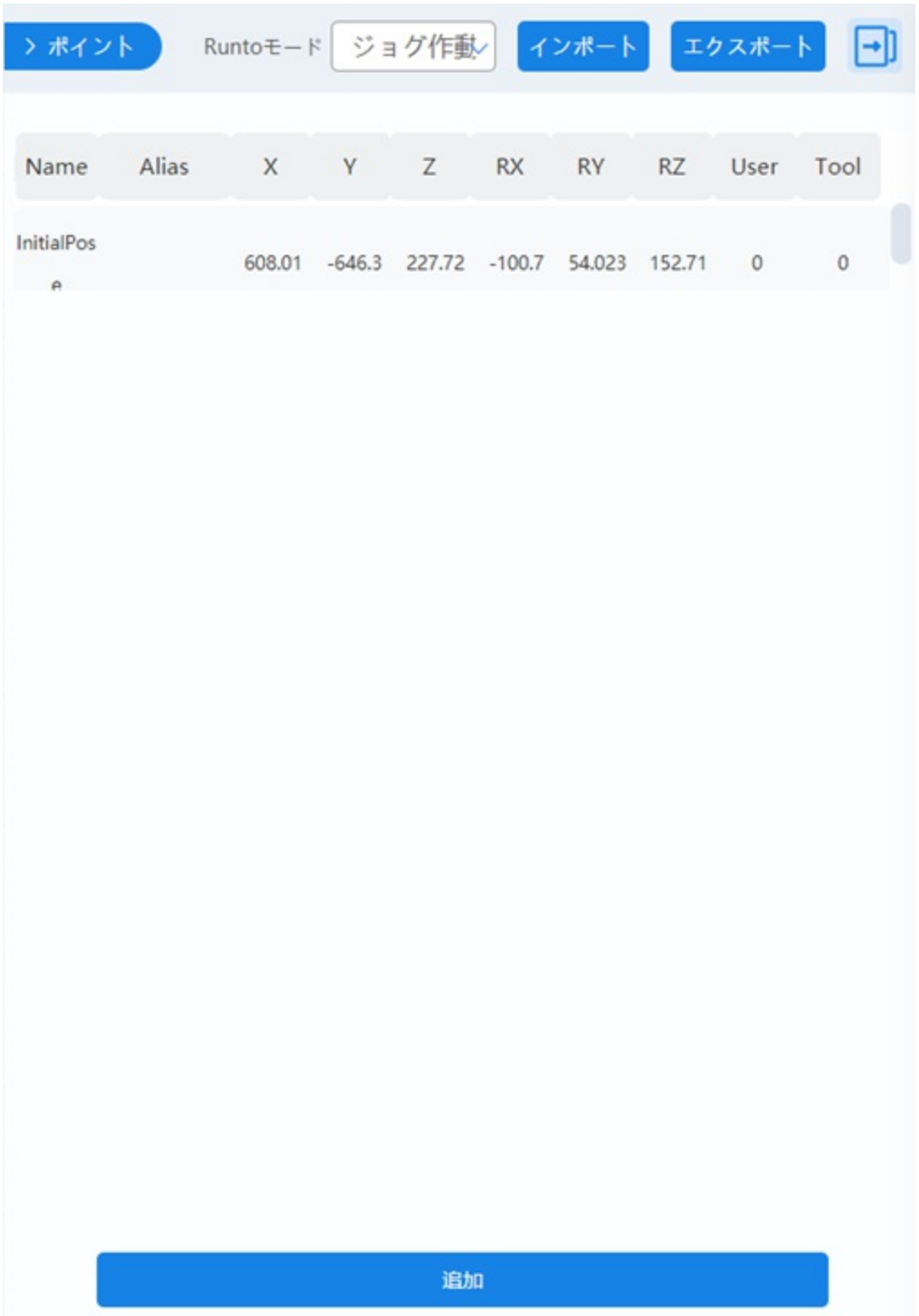
3. 制御パネルで前に設定したツール座標系を選択します。

プロジェクトを作成する

プログラミングの詳細な説明はグラフィカルプログラミングとスクリプトプログラミングを参照してください。ここではグラフィカルプログラミングを例に簡単に説明します。

本章の冒頭に説明したサンプルシナリオを実現するため、まず、それぞれ物品ピック点P1、物品ピック点上部の遷移点P2、積み下ろし点上部の遷移点P3、および積み下ろし点P4の4つの点をティーチングする必要があります。

1. 点保存のページを開き、ロボットアームをP1まで移動した後、「新規追加」をクリックします。



1. 同様の方法で、P2、P3、P4のティーチングを追加します。
2. ブロックプログラミングをドラッグして、ピークと積み下ろしを実現します。簡単なプログラムの例は以下のとおりです。

3. プロジェクトを保存します。

リモートI/Oモードを構成し、ログインする

リモート制御に関する詳細な説明はリモート制御の章節を参照してください。ここではサンプルシナリオに基づいた簡単な説明を行うだけです。

1. 「設定 > リモート制御」ページを開きます。
2. 現在のモードを「リモートI/Oモード」に変更します。
3. 「開く」をクリックし、前段階で保存したグラフィカルプログラミングのプロジェクトを選択します。
4. 「変更」をクリックし、本章の冒頭で説明したシナリオに基づきI/Oの構成を変更します。
5. 「適用」をクリックし、リモートI/Oモードに進みます。

リモートI/Oに進んだ後、ロボットアームのコントロールキャビネットに接続した開始ボタンを押すと、ロボットアームはプロジェクト実行を開始します。

付属書A Modbusレジスタの定義

1 Modbusの紹介

Modbusはシリアル通信プロトコルです。このプロトコルにより、CRシリーズのロボットは外部設備と通信することができます。外部設備、例えばPLCを通じてCRシリーズのロボットを制御する場合、外部設備はModbusマスターとして、CRシリーズロボットはModbusスレーブとなります。

Modbusは現在、以下のバージョンがあります。

- Modbus-RTU：コンパクトなバイナリのデータ表示方式を採用します。巡回冗長検査を使用したチェックサム方式です。
- Modbus-ASCII：ASCIIコードの文字列を基にした表示方式で、読み取り可能で、冗長な表示方式です。縦方向の冗長性チェックを使用したチェックサム方式です。
- Modbus-TCP：TCP/IPのTCP方式により接続します。これらの方式はチェックサムの計算は不要です。

当社の既存のハードウェアインターフェスを基にして、現在、Modbus-TCPを採用しています。

通常、Modbusアドレスは5桁の数字で構成されています。開始データタイプコード、その後のオフセットアドレスを含みます。Modbus Masterプロトコルライブラリは、標準的なModbusアドレスをModbus機能番号にマッピングし、スレーブのデータを読み書きします。

Modbus Masterプロトコルライブラリは次のアドレスをサポートします。

- 00001-09999：デジタル出力（コイル）
- 10001-19999：デジタル入力（接点）
- 30001-39999：入力データレジスタ（通常はアナログ入力）
- 40001-49999：データ保持レジスタ

Modbusプロトコルのデータは主に4種類に分けられ、コイル状態、離散入力、入力レジスタ、保持レジスタです。CRシリーズのロボットのメモリースペースを基にして、当社はコイル、接点（離散入力）、入力、保持レジスタを定義し、外部設備とロボットの制御システムがデータを交換するために使用します。ただし、各レジスタのアドレスは4096個です。詳細説明は以下のとおりです。

2 コイルレジスタ（ロボット制御）

PLCアドレス	スクリプトアドレス	レジスタタイ	機能

PLCアドレス	（Get/SetCoils）	プ	機能
00001	0	Bit	開始
00002	1	Bit	一時停止
00003	2	Bit	継続
00004	3	Bit	停止
00005	4	Bit	緊急停止
00006	5	Bit	アラームのクリア
00007	6	Bit	電源オン
00101	100	Bit	イネーブル
00102	101	Bit	ディセーブル
03096~10000	3095~9999	Bit	カスタム

3 離散入力の定義（ロボットの状態）

PLCアドレス	スクリプトアドレス (GetInBits)	レジスタタイプ	機能
10002	1	Bit	停止状態
10003	2	Bit	一時停止中
10004	3	Bit	実行状況
10005	4	Bit	アラーム状態
10006	5	Bit	システム予約済み
10008	7	Bit	リモートモード
13096~20000	3095~9999	Bit	カスタム

4 入力レジスタの定義

PLCアドレス	データタイプ	値の数	バイト数	スクリプトアドレス (GetInRegs)	レジスタタイプ	機能
	unsigned					

31001	unsigned short	1	2	1000	U16	メッセージバイトの総長さ
31002~31004	unsigned short	3	6	1001~1003	U16	予約ビット
31005~31008	uint64	1	8	1004~1007	U64	デジタル入力
31009~31012	uint64	1	8	1008~1011	U64	デジタル出力
31013~31016	uint64	1	8	1012~1015	U64	ロボットモード
31017~31020	uint64	1	8	1016~1019	U64	タイムスタンプ
31021~31024	uint64	1	8	1020~1023	U64	予約ビット
31025~31028	uint64	1	8	1024~1027	U64	メモリ構造テスト基準値 0x0123 4567 89AB CDEF
31029~31032	double	1	8	1028~1031	F64	予約ビット
31033~31036	double	1	8	1032~1035	F64	速度比
31037~31040	double	1	8	1036~1039	F64	ロボットの現在の運動量
31041~31044	double	1	8	1040~1043	F64	制御パネルの電圧（CCBOXではサポートされていません）
31045~31048	double	1	8	1044~1047	F64	ロボット電圧（CCBOXではサポートされていません）
31049~31052	double	1	8	1048~1051	F64	ロボット電流（CBOXではサポートされていません）
31053~31056	double	1	8	1052~1055	F64	予約ビット
31057~31060	double	1	8	1056~1059	F64	予約ビット
31061~31072	double	3	24	1060~1071	F64	予約ビット
31073~31084	double	3	24	1072~1083	F64	予約ビット
31085~31096	double	3	24	1084~1095	F64	予約ビット
31097~31120	double	6	48	1096~1119	F64	目標関節位置
31121~31144	double	6	48	1120~1143	F64	目標関節速度
31145~31168	double	6	48	1144~1167	F64	目標関節加速度
31169~31192	double	6	48	1168~1191	F64	目標関節電流
31193~31216	double	6	48	1192~1215	F64	目標関節トルク

31193~31216	double	6	48	1192~1215	F64	目標関節トルク
31217~31240	double	6	48	1216~1239	F64	実際の関節位置
31241~31264	double	6	48	1240~1263	F64	実際の関節速度
31265~31288	double	6	48	1264~1287	F64	実際の関節電流
31289~31312	double	6	48	1288~1311	F64	TCPセンサー力値（6次元力センサーの設置が必要）
31313~31336	double	6	48	1312~1335	F64	TCPの実際のデカルト座標値
31337~31360	double	6	48	1336~1359	F64	TCPの実際のデカルト速度値
31361~31384	double	6	48	1360~1383	F64	TCP力値（関節電流により計算）
31384~31408	double	6	48	1384~1407	F64	TCPの目標のデカルト座標値
31409~31432	double	6	48	1408~1431	F64	TCPの目標のデカルト速度値
31433~31456	double	6	48	1432~1455	F64	関節温度
31456~31480	double	6	48	1456~1479	F64	関節コントロールモード
31481~31504	double	6	48	1480~1503	F64	関節電圧
31505~31506	char	4	4	1504~1505	U16	ハンドシステム
31507	char	1	1	1506下位バイト	U8	ユーザー座標
31507	char	1	1	1506上位バイト	U8	工具座標
31508	char	1	1	1507下位バイト	U8	アルゴリズムキュー実行フラグ
31508	char	1	1	1507上位バイト	U8	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	U8	関節速度比率
31509	char	1	1	1508上位バイト	U8	関節加速度比率
31510	char	1	1	1509下位バイト	U8	関節加加速度比率
31510	char	1	1	1509上位バイト	U8	デカルト位置速度比率
31511	char	1	1	1510下位バイト	U8	デカルト姿勢速度比率

31511	char	1	1	1510上位バイト	U8	デカルト位置加速度比率
31512	char	1	1	1511下位バイト	U8	デカルト姿勢加速度比率
31512	char	1	1	1511上位バイト	U8	デカルト位置加加速度比率
31513	char	1	1	1512下位バイト	U8	デカルト姿勢加加速度比率
31513	char	1	1	1512上位バイト	U8	ロボットのブレーキ状態
31514	char	1	1	1513下位バイト	U8	ロボットのイネーブル状態
31514	char	1	1	1513上位バイト	U8	ロボットのドラッグ状態
31515	char	1	1	1514下位バイト	U8	ロボットの動作状態
31515	char	1	1	1514上位バイト	U8	ロボットのアラーム状態
31516	char	1	1	1515下位バイト	U8	ロボットのジョグ状態
31516	char	1	1	1515上位バイト	U8	ロボットのタイプ
31517	char	1	1	1516下位バイト	U8	ボタンパネルドラッグ信号
31517	char	1	1	1516上位バイト	U8	ボタンパネルイネーブル信号（Novaシリーズではサポートされていません）
31518	char	1	1	1517下位バイト	U8	ボタンパネル記録信号（Novaシリーズではサポートされていません）
31518	char	1	1	1517上位バイト	U8	ボタンパネル再現信号（Novaシリーズではサポートされていません）
31519	char	1	1	1518下位バイト	U8	ボタンパネルグリッパー制御信号（Novaシリーズではサポートされていません）
31519	char	1	1	1518上位バ	U8	6次元力オンライン状態（6次元力センサーの設置が

				イト		センサーの設置が必要）
31520	char	1	1	1519下位バイト	I8	衝突状態
31520	char	1	1	1519上位バイト	I8	(セーフティスキン) 前腕が一時停止状態に近づいています
31521	char	1	1	1520下位バイト	I8	(セーフティスキン) J4は一時停止状態に近づいています
31521	char	1	1	1520上位バイト	I8	(セーフティスキン) J5は一時停止状態に近づいています
31522	char	1	1	1521下位バイト	I8	(セーフティスキン) J6は一時停止状態に近づいています
31522	char	1	1	1521上位バイト	I8	予約ビット
31523~31552	-	-	-	1522~1551	-	予約ビット
31553~31556	double	1	8	1552~1555	F64	予約ビット
31557~31560	uint64	1	8	1556~1559	U64	予約ビット
31561~31584	double	6	48	1560~1583	F64	実際のトルク
31585~31588	double	1	8	1584~1587	F64	負荷重量(kg)
31589~31592	double	1	8	1588~1591	F64	X方向偏心距離(mm)
31593~31596	double	1	8	1592~1595	F64	Y方向偏心距離(mm)
31597~31600	double	1	8	1596~1599	F64	Z方向偏心距離(mm)
31601~31624	double	6	48	1600~1623	F64	ユーザー座標値
31625~31648	double	6	48	1624~1647	F64	工具座標値
31649~31652	double	1	8	1648~1651	F64	軌跡再現運転インデックス
31653~31676	double	6	48	1652~1675	F64	現在の6次元力データの元の値（6次元力センサーの設置が必要）
						[qw、qx、qy、qz]

						目標の四元数
31693~31708	double	4	32	1692~1707	F64	[qw、qx、qy、qz] 実際の四元数
31709~31721	double	1	24	1708~1720	F64	予約ビット
			1440			合計1440バイト

5 保持レジスタの定義（ロボットPLC対話）

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
40101	100	U16	グローバル速度を設定します。値の範囲は1〜100で、値の範囲を超える値を書き込んでも有効になりません。
40001~41281	0~1280	U16	パレタイジング用に予約されています。
43095~49000	3095~8999	U16	カスタム
49001	9000	U16	現時点のねじシリアル番号
49002	9001	U16	実行するねじ番号を指定： 番号は1から始まり、0はねじ番号の指定がないことを示す
49003	9002	U16	ねじロック装着の結果： 0は失敗、1は成功を示す。2はNoneを示す
49004	9003	U16	製品（ステーション）上にロックしたねじの数
49005~49054	9004~9053	U16	製品ロック装着の結果： 最大50組をサポート。0は失敗、1は成功を示す。2はNoneを示す
49055	9054	U16	アラームコード： 0はアラームなし。1はピックアップ時のねじマシンの材料。2はピックアップ前にヘッドにねじあり。3はねじピックアップ失敗。4はねじロック装着異常。5は電動ドライバー異常
49056	9055	U16	ロック装着するねじの総数
49057	9056	U16	ロック装着する不良ねじの総数
49058	9057	U16	ねじロック装着の製品総数
49201	9220	F64	ねじロック装着のトルク（Nm)

49201	9220	F64	ねじロック装着のトルク（Nm)
49205	9224	F64	ねじロック装着の角度（°）
49209	9228	F64	ねじロック装着のタクト（s）
49213~49412	9232~9431	F64	製品ロック装着のトルク（Nm）。 最大50組をサポート
49413~49612	9432~9631	F64	製品ロック装着の角度（°） 最大50組をサポート

付属書B グラフィカルプログラミングブロックの説明

クイックエクスペリエンス

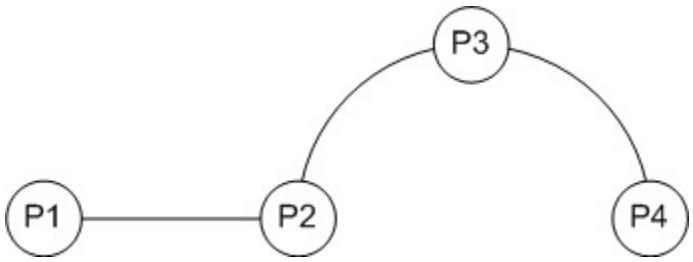
- ロボットアームの移動制御
- **Modbus**レジスタデータの読み取り/書き込み
- **TCP**通信によるデータの転送
- パレットブロックを使用したパレタイジング

ロボット動作の制御

シーンの説明

グラフィカルプログラミングでロボットの移動を制御する方法を体験するために、まず以下のシーンの実現を仮定します。

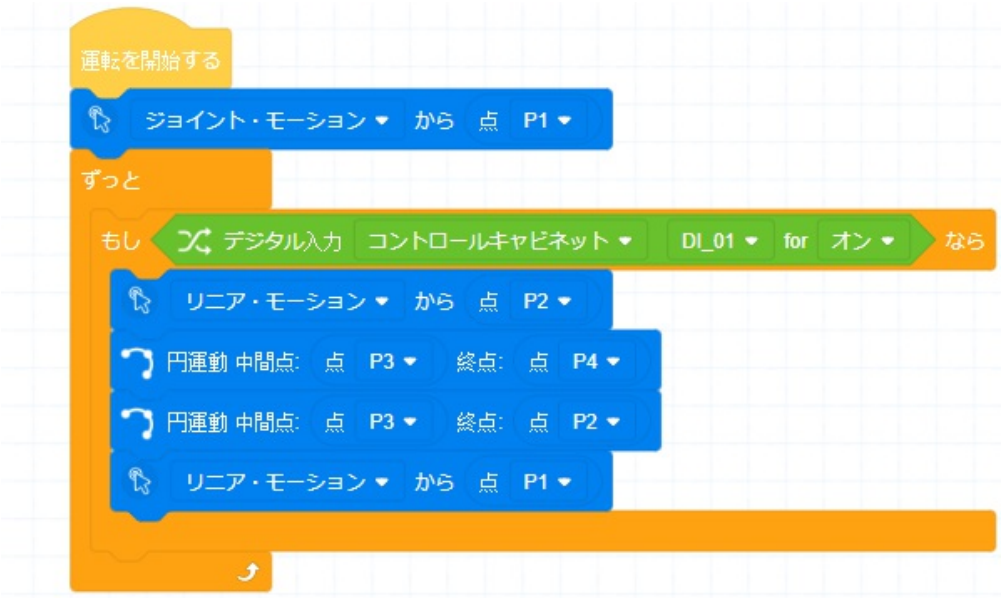
制御盤DI1がONである場合、ロボットはP1からP2に直線移動した後、P3を経てP4に円弧移動し、さらに元の経路で戻ります。制御盤DI1がOFFである場合、ロボットアームは移動しません。



上手のティーチングP1~P4を参考にしてください。

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. ロボットは起点(P1)まで関節移動します。
2. 無条件ループを設定すると、後のコマンドはプログラム実行時にループし続けます。
3. 制御盤DI1がONであるかを判断し、ONである場合に後続のプログラムを実行します。そうでない場合、直接次のループに進み、DI1の状態を再度取得します。
4. ロボットはP2まで直線移動します。

5. ロボットはP3を経てP4まで円弧移動します。
6. ロボットはP3を経てP2まで円弧移動します。
7. ロボットはP1に直線移動し、次のループに入り、ステップ3に戻ります。

プログラムの実行

点位置のティーチングとプログラミングが完了すれば、プログラムを実行するだけです。DI1の状態は、IOモニタリングパネル中の仮想DIにより設定します。

Modbusレジスタデータの読み取り/書き込み

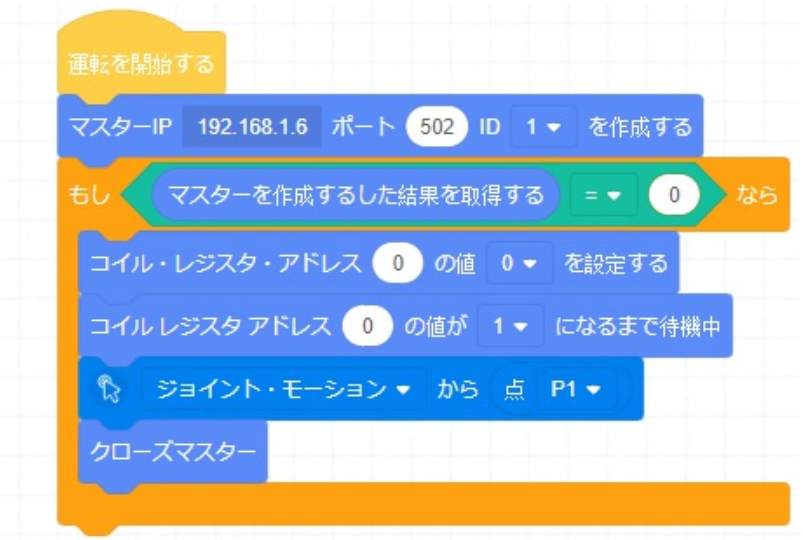
シーンの説明

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

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

プログラミング手順

上記シーンを実現するため、作成するプログラムは下図のとおりです。



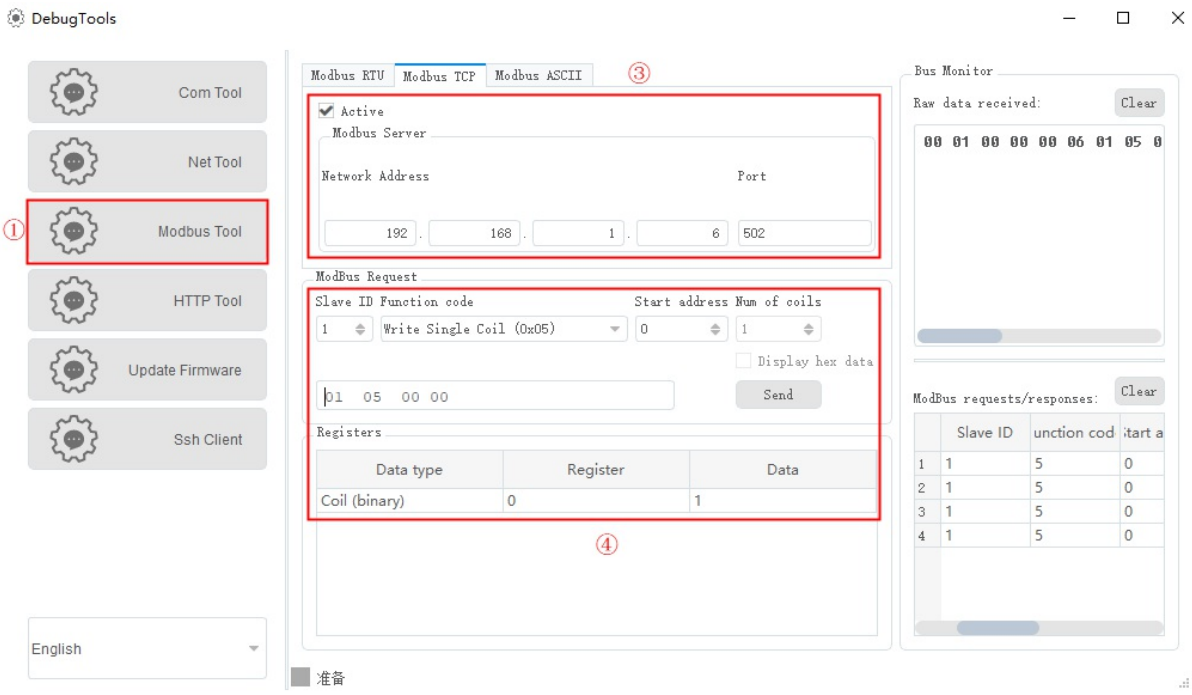
1. マスターを作成し、IPアドレスはスレーブアドレスで、ポートとIOはデフォルト値です。ここで迅速に検証するため、ロボット自身のスレーブを使用します。そのため、IPアドレスはロボットのアドレスです。
2. マスターの作成が完了したかを判断し、作成完了していると後続のステップを実行できます。そうでない場合、プログラムは終了します。
3. ロボットコイルレジスタ0の値が修正されると、後続プログラムの論理に影響を及ぼす可能性があります。そのためまず、コイルレジスタ0の値を0に設定します。
4. コイルレジスタの値が1に変わります。
5. ロボットを制御してP1点まで移動します。P1はユーザ定義の点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

プロジェクトの実行

迅速にこのプログラムを実行する必要がある場合、DobotStudio Proのデバッグツールを使用してコイルレジスタの値を修正できます。

1. デバッグツールを開き、「Modbusデバッグツール > Modbus TCP」画面に入ります。
2. ロボットはP1以外の実行点に移動し（ロボットが移動コマンドを実行したかを後で確認しやすい）、プログラムを保存、実行します。
3. 実行ログがModbusマスター作成完了を表示した後、デバッグツールで「有効化」をチェックし、Modbusサーバのネットワークアドレスとポートを修正します。
4. Modbusが求める機能コードを「単一コイル書き込み」に変更し、レジスタ0のデータを1に変更し、再度「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。

下図はデバッグツール画面で、図中の番号は上記ステップに対応しています。


DebugTools
Com Tool
Net Tool
①
Modbus Tool
HTTP Tool
Update Firmware
Ssh Client
English
Modbus RTU
Modbus TCP
Modbus ASCII
③
Active
Modbus Server
Network Address
Port
192
168
1
6
502
ModBus Request
Slave ID
Function code
Start address
Num of coils
1
Write Single Coil (0x05)
0
1
Display hex data
01 05 00 00
Send
Registers
Data type
Register
Data
Coil (binary)
0
1
④
Bus Monitor
Raw data received:
Clear
00 01 00 00 00 06 01 05 0
ModBus requests/responses:
Clear
Slave ID
unction cod
tart a
1 1 5 0
2 1 5 0
3 1 5 0
4 1 5 0
准备

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。

プログラムの実行

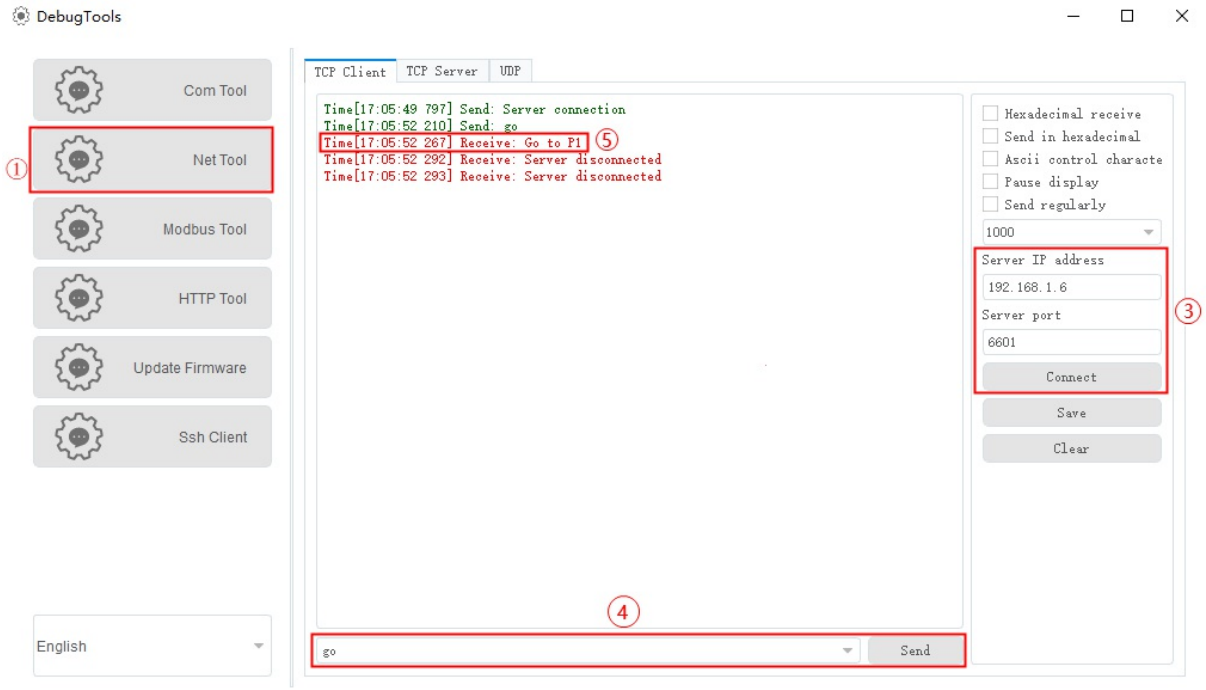
迅速にこのプログラムを実行する必要がある場合、DobotStudio Proのデバッグツールを TCPクライアントとして使用できます。

1. デバッグツールを開き、「ネットワークデバッグツール > TCPクライアント」画面に進み

ます。

2. ロボットはP1以外の実行点に移動し（ロボットが移動コマンドを実行したかを後で確認しやすい）、プログラムを保存、実行します。
3. 実行ログがTCPサーバ作成完了を表示した後、デバッグツールでサーバのIPアドレスとポートを修正し、接続をクリックします。
4. 接続できた場合、デバッグツール下で「go」を入力して送信をクリックします。
5. デバッグツールが「Go to P1」情報を受信し、ロボットがP1まで移動するかを確認します。

下図はデバッグツール画面で、図中の番号は上記ステップに対応しています。

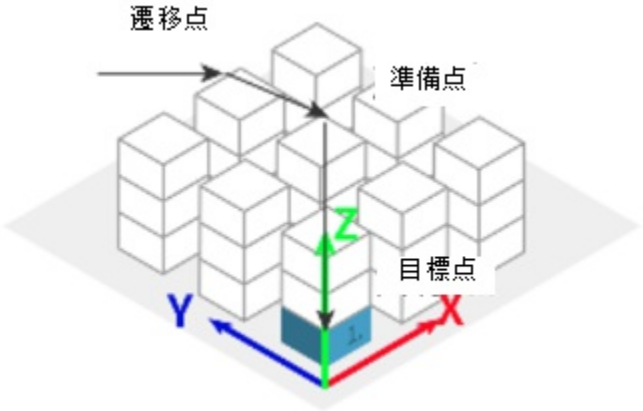


パレットブロックを使用したパレタイジング

シーンの説明

運搬応用では、一部の運搬対象の物品は規則的に並べられ、間隔も均等ですので、各物品の位置を一々ティーチングするのは誤さが大きく、 効率が低いという問題があります。パレットブロックを使って積み重ねパターンを設定し、各物品の位置を自動的に算出することにより、これらの問題を効率的に解決することができます。

まず、物品を立方体に積み重ねる必要があると仮定し、まず1つの目標の積み重ねパターンを手動でパレタイジングし、次に関連点位置をティーチングする必要があります。



- 遷移点はP1であるとティーチングします。遷移点はロボットアームがパレタイジングエリアに進出する際にいずれも通過する点位置であり、安全 遷移のためのもので、物品ピック点の上方であるとティーチングできます。
- 物品ピック箱の点位置はP2であるとティーチングします。
- 準備点と目標点は個別に逐次ティーチングする必要がありません。後続の積み重ねパターンの設定を参照してください。

次に、ロボットアームの先端にグリッパーや吸盤などの機構が取り付けられており、コントロールキャビネットのDO1で物品のピックとプレースを制御すると仮定します。

積み重ねパターンの設定

作成したパレットブロックをプログラミングエリアにドラッグし、ブロックをクリックするとパレットパネルが開きます。

パレットの次元

- 1次元：物品が一行に並べられ、物品総数がX方向における個数に等しい。
- 2次元：物品が方形に並べられ、物品総数がX方向とY方向における個数の積に等しい。
- 3次元：物品が立方体に積み重ねられ、物品総数が3つの方向における個数の積に等しい。

本マニュアルは3次元積み重ねパターンを例とし、各方向における物品個数を10とすると、1つの完全な積み重ねパターンには1000個の物品があります。

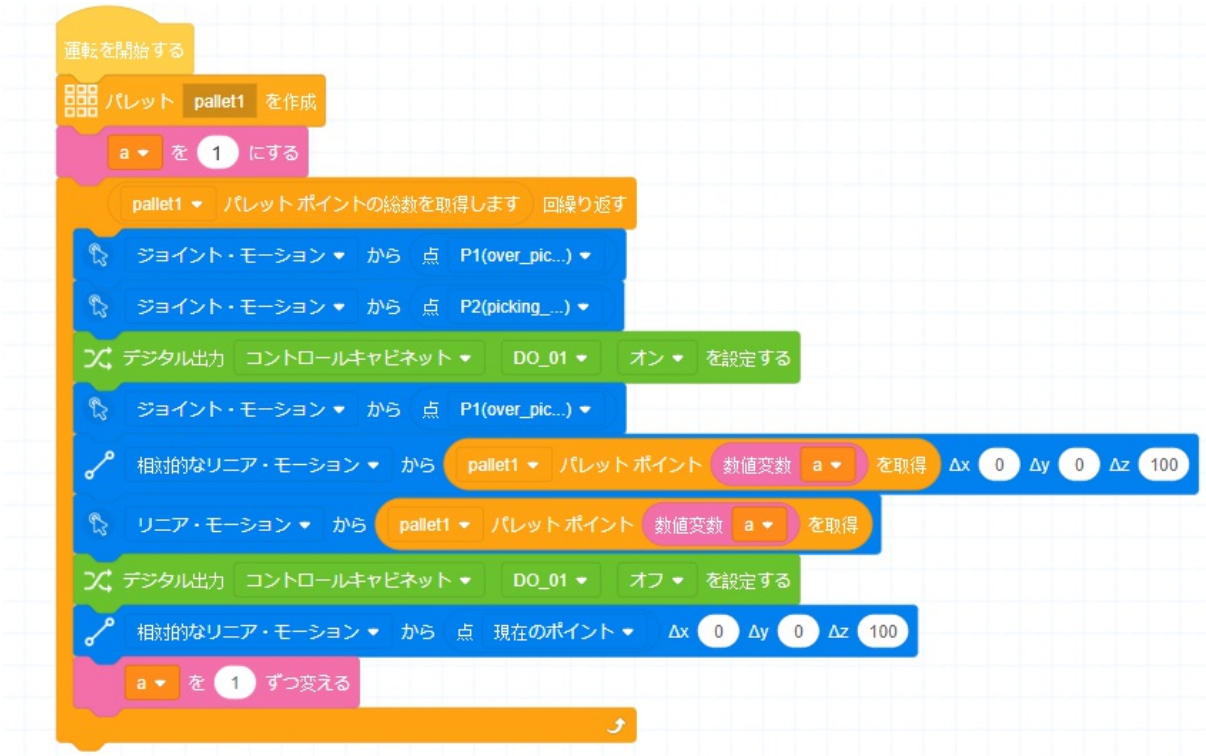
点位置の設定

3次元積み重ねパターンを例とすると、ここでは8つの点を配置する必要があり、それぞれ立方体の8つの角上の物品の目標点（物品をプレースするときのロボットアームの点位置）に対応し、制御システムはこれらの8つの点と物品の数量によって各物品の目標点を自動的に算出し、X->Y->Z座標軸の順番で並べ替えます。

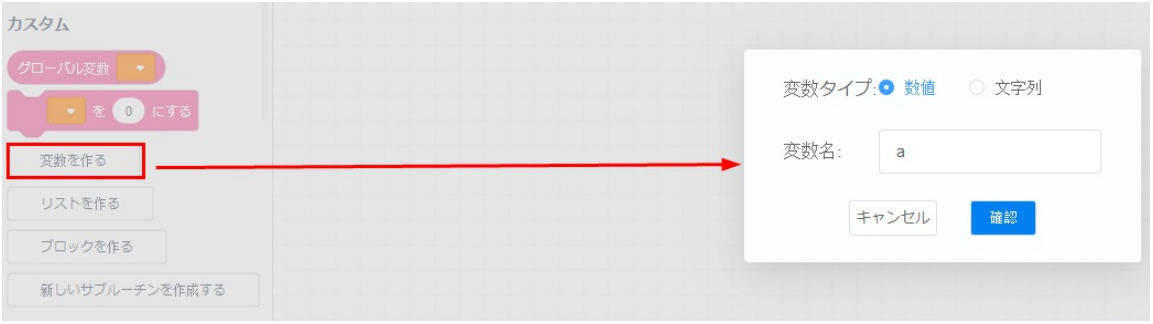
点位置を設定するときには、プロジェクトにおけるティーチング済みの既存点を選択してもよく、「カスタマイズ」をクリックして現在の点位置を取得してもよい。設定済みの点位置のアイコンは緑色に変わります。

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. パレットpallet1を作成します。
2. カスタム数値の変数を新たに作成し、値を1に設定し、ループ回数の記録に使用します。



3. 後続のコードをループで実行し、回数は対応するパレットの総数です。
4. ロボットはピックアップ点上方(P1)まで移動します。
5. ピックアップ点(P2)まで移動します。
6. DO1を開で設定し、グリッパーを制御して物品をピックアップします。
7. ロボットはピックアップ点上方(P1)まで戻ります。
8. 現在のパレット点上方100mmの位置まで移動します。
9. 現在のパレット点まで移動します。
10. DO1を閉で設定し、グリッパーを制御して物品をプレースします。
11. 現在のパレット点上方100mmの位置まで戻ります。
12. ループ回数に1を加算し、ループはステップ4まで戻ります。

本マニュアルで書いたプログラムは簡単な例に過ぎず、実際の応用では実際の状況にさらなるI/O制御や状況を加味して判断することができます。たとえば、物品をピックしていないときは後続の動作を実行しない。

プログラムの実行

点位置のティーチング、積み重ねパターン、プログラミングが完了すれば、プログラムを実行するだけです。DO1の状態は、IOモニタリングパネル中で確認できます。

ブロックの説明

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、プログラム実行開始としての標識に使用されます。

実行開始

説明：プログラムのメインスレッドの標識です。新しいプロジェクトを作成した後に、プログラミングエリアには実行開始標識があります。その他非イベントブロックをその下に接続し、プログラミングを行ってください。

使用制限：1つのプロジェクトに「実行開始」ブロックは1つのみです。

サブスレッド起動

説明：プログラムのサブスレッドの標識です。サブスレッドはメインスレッドと同期して実行します。ただし、サブスレッドはロボットアームの制御コマンドを使用できず、変数演算またはI/O制御などのみを実行できます。プログラム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：1つのプロジェクト中に使用できるサブスレッドは最大5個です。

制御ブロックグループ

制御ブロックグループはプログラムの実行経路を制御するために使用されます。

条件を満たすまで待機

説明：プログラムはパラメータがtrueになって実行を続けるまで一時停止します。

パラメータ：その他のヘキサゴンブロックを使用してパラメータとします。

n回繰り返し実行

説明：その他のブロックをこのブロックの中間にネストにすると、ネストにされたブロックのインストラクションが指定された回数繰り返し実行されます。

パラメータ：繰り返し実行する回数。

継続して繰り返し実行

説明：その他のブロックをこのブロックの中間にネストにすると、ネストにされたブロックのインストラクションが繰り返し終了ブロックに遭遇するまでずっと繰り返し実行されます。

繰り返し終了

説明：繰り返し実行クラスのブロックにネストするために使用され、プログラムがこの基本まで実行された時、繰り返しを直接終了し、ブロックを繰り返した後のブロックインストラクションを実行します。

条件を満たした後に実行

説明：パラメータがtrueの場合、ネストされたブロックインストラクションを実行します。パラメータがfalseの場合、次のブロックインストラクションに直接スキップします。

パラメータ：その他のヘキサゴンブロックを使用して（戻り値がブール値、即ちtrueまたはfalse）パラメータとします。

条件を満たしたまたは満たさなかった後にそれぞれ実行

説明：パラメータがtrueの場合、「else」の前のネストにされたブロックインストラクションを実行します。パラメータがfalseの時、「else」の後のネストにされたブロックインストラクションを実行します。

パラメータ：その他のヘキサゴンブロックを使用して（戻り値がブール値、即ちtrueまたはfalse）パラメータとします。

条件を満たすまで繰り返し実行

説明：ネストされたブロックインストラクションを、パラメータがtrueになるまで繰り返し実行します。

パラメータ：その他のヘキサゴンブロックを使用して（戻り値がブール値、即ちtrueまたはfalse）パラメータとします。

タグの設定

説明：タグを設定します。設定後、タグジャンプブロックによりジャンプができます。

パラメータ：タグの名称で、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。

タグのジャンプ

説明：プログラムがこのブロックまで実行されると、指定のタグまで直接ジャンプし、タグの後のブロックインストラクションを実行します。

パラメータ：設定済みのタグ名称。

インストラクションの折りたたみ

説明：ネストされたブロックを折りたたんでで表示できます。制御の役割は果たさず、ただプログラムをさらに美しく読みやすくするために使用されます。

パラメータ：折りたたまれたブロックを説明するには、間接的で直感的な名称をつけることをお勧めします。

一時停止

説明：プログラムはこのブロックまで実行した後、自動的に一時停止します。制御ソフトウェアまたはリモート制御により実行を継続できます。

衝突検出機能の設定

説明：衝突検出機能を設定します。このブロックを介して設定した衝突検出等級は、プロジェクト実行期間にのみ有効で、プロジェクト停止した後に変更前の値に戻ります。

パラメータ：衝突検出機能の感度を選択し、オフまたは等級1～5を選択することが可能です。等級の数字が大きいほど衝突検出の感度も高くなります。

ユーザー座標系の変更

説明：指定されたユーザー座標系を変更します。この変更は現在のプロジェクト実行中にのみ有効で、プロジェクトが停止後、座標系は変更前の値に戻ります。

パラメータ：

- 変更するユーザー座標系の番号を指定します。
- 変更後のユーザー座標系のパラメータを指定します。

ツール座標系の変更

説明：指定されたツール座標系を変更します。この変更は現在のプロジェクト実行中にのみ有効で、プロジェクトが停止後、座標系は変更前の値に戻ります。

パラメータ：

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

パレットの作成

説明：パレットの積み重ねパターンを作成するために使用され、詳細はパレットブロックを使用したパレタイジングを参照してください。

パラメータ：作成するパレットの名称。

指定したパレットの合計数の取得

説明：指定したパレットのターゲットスタックの総数を取得します。

パラメータ：パレットの名称。

指定したパレット点位置座標系の取得

説明：指定したパレットの指定点位置座標系を取得します。

パラメータ：

- パレットの名称。
- 点位置の索引番号で、1から始まります。

インストラクション実行の遅延

説明：プログラムがこのブロックまで実行された後、指定された時間一時停止してから、再度継続して実行されます。

パラメータ：プログラムが一時停止する時間。

移動待機インストラクション

説明：移動ブロックの前後に接続され、移動インストラクションを遅延させたり、移動インストラクション完了後に次のインストラクションを遅延させたりする作用を果たします。

パラメータ：インストラクションを遅延する時間。

システム時間の取得

説明：システムの現在の時間を取得します。

戻り値：システムの現在の時間のUnixタイムスタンプ。

演算ブロックグループ

演算ブロックグループは変数または定数を演算するために使用されます。

四則演算

説明：パラメータを四則演算します。

パラメータ：

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

戻り値：演算の結果の数値。

比較演算

説明：パラメータを比較演算します。

パラメータ：

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

戻り値：比較結果が真の時trueを返し、偽の時falseを返します。

AND演算

説明：パラメータをAND演算します。

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

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

OR演算

説明：パラメータをOR演算します。

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

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

NOT演算

説明：パラメータをNOT演算します。

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

戻り値：パラメータが真の時falseを返し、偽の時trueを返します。

剰余演算

説明：パラメータを剰余演算します。

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

戻り値：演算の結果の数値。

四捨五入演算

説明：パラメータを四捨五入演算します。

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

戻り値：演算の結果の数値。

単項演算

説明：パラメータについてさまざまな単項演算をします。

パラメータ：

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

戻り値：演算の結果の数値。

プリント

説明：パラメータをコンソールに出力して確認し、主にデバッグに使用されます。

パラメータ：

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

文字ブロックグループ

文字ブロックグループには、文字列とアレイの一般機能が含まれています。

文字列のn番目の文字の取得

説明：変数とする文字列のn番目の文字を取得します。

パラメータ：

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

戻り値：文字列の指定位置の文字。

文字列1が文字列2を含むか否かの判断

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

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

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

2つの文字列の連結

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

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

戻り値：連結後の文字列。

文字列またはアレーの長さの取得

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

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

戻り値：文字列またはアレーの長さの数値。

2つの文字列の比較

説明：ACSIIコードに基づき、2つの文字列の大きさを比較します。

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

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

アレーの文字列への変換

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

パラメータ：

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

戻り値：変換後の文字列。

文字列のアレーへの変換

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

パラメータ：

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

戻り値：変換後のアレー。

アレーのインデックスを指定する要素の取得

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

パラメータ：

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

戻り値：アレーの位置を指定する要素の値。

アレー内で指定した複数の要素の取得

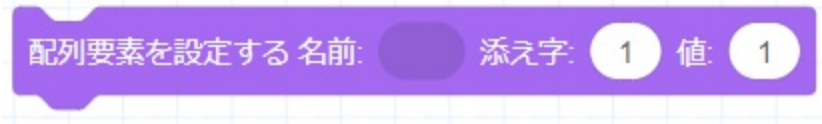
説明：指定するアレー内の複数のインデックス位置を指定する要素を取得します。最初のインデックスと最後のインデックスの範囲内で、ステップ値に基づき要素を取得します。

パラメータ：

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

戻り値：指定する元素が構成する新しいアレー。

アレー中の指定された要素の設定



説明：アレー中のインデックス位置を指定する要素の値を設定します。

パラメータ：

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

カスタムブロックグループ

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

グローバル変数の呼び出し

説明：制御ソフトウェアで設定されるグローバル変数を呼び出します。

パラメータ：グローバル変数の名称を選択します。

戻り値：グローバル変数の値。

グローバル変数の設定

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

パラメータ：

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

カスタム変数の新規作成

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

カスタム数値変数

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

戻り値：変数の値。

カスタム数値の変数の値設定

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

パラメータ：

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

カスタム数値の変数の値増減

説明：指定した数値の変数を、指定した値分で増やします。

パラメータ：

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

カスタム文字列変数

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

戻り値：変数の値。

カスタム文字列の変数の値設定

説明：指定された文字列の変数を設定します。

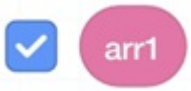
パラメータ：

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

アレーの作成

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

カスタムアレー



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

戻り値：アレーの値。

変数のアレーへの追加

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

パラメータ：

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

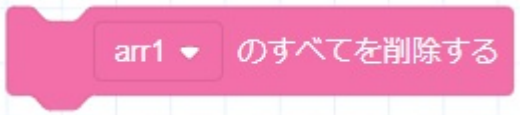
アレーの指定されたアイテムの削除

説明：指定されたアレーの指定されたアイテムを削除します。

パラメータ：

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

アレーのすべてのアイテムの削除



説明：指定されたアレーのすべてのアイテムを削除します。

パラメータ：変更するアレーを選択します。

アレーへのアイテムの挿入

説明：アレー中の指定された位置に変数を挿入します。

パラメータ：

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

アレーにおける指定されたアイテムの置換

説明：アレー中の指定されたアイテムを指定された変数に置換します。

パラメータ：

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

アレーにおける指定されたアイテムの取得

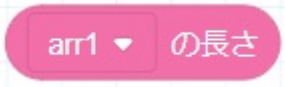
説明：アレー中の指定されたアイテムの値を取得します。

パラメータ：

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

戻り値：指定されたアイテムの値。

アレーにおける総アイテム数の取得



説明：アレー中のアイテムの総数を取得します。

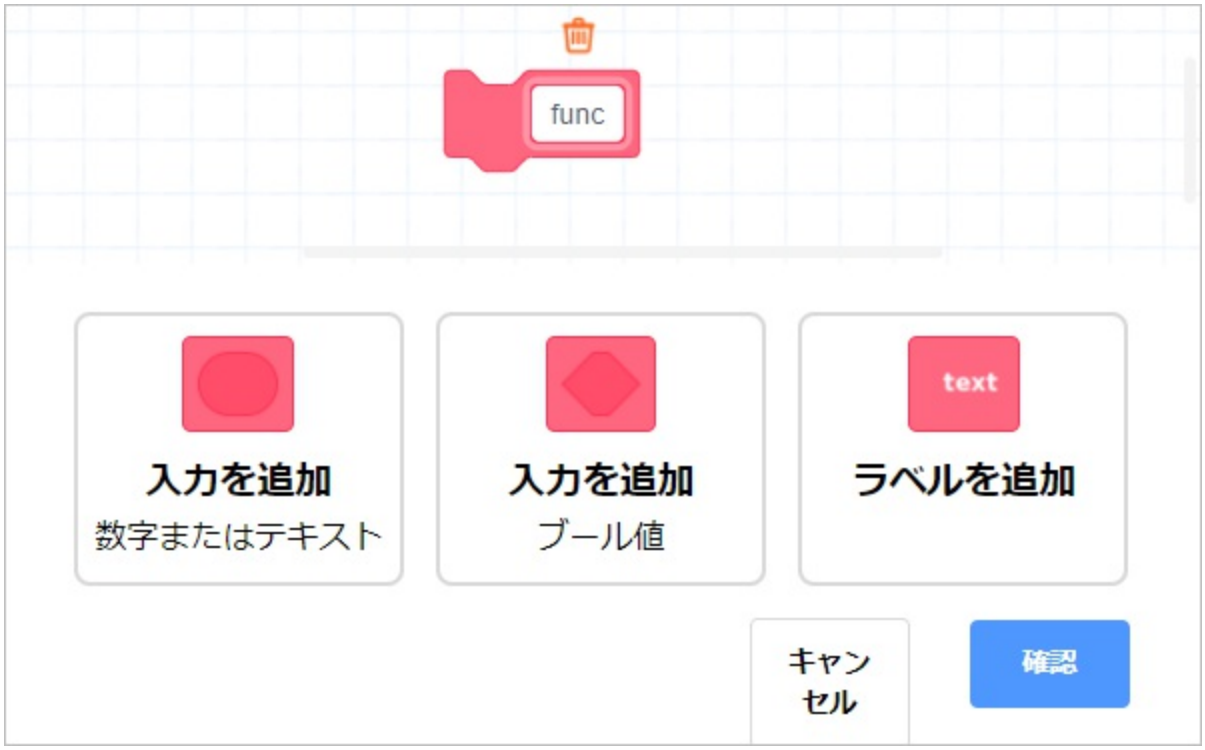
パラメータ：取得するアイテムのアレーを選択します。

戻り値：指定するアレーのアイテム総数。

1つの関数の新規作成

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

1. 関数の宣言



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

1. 関数の定義

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

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

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

カスタム関数

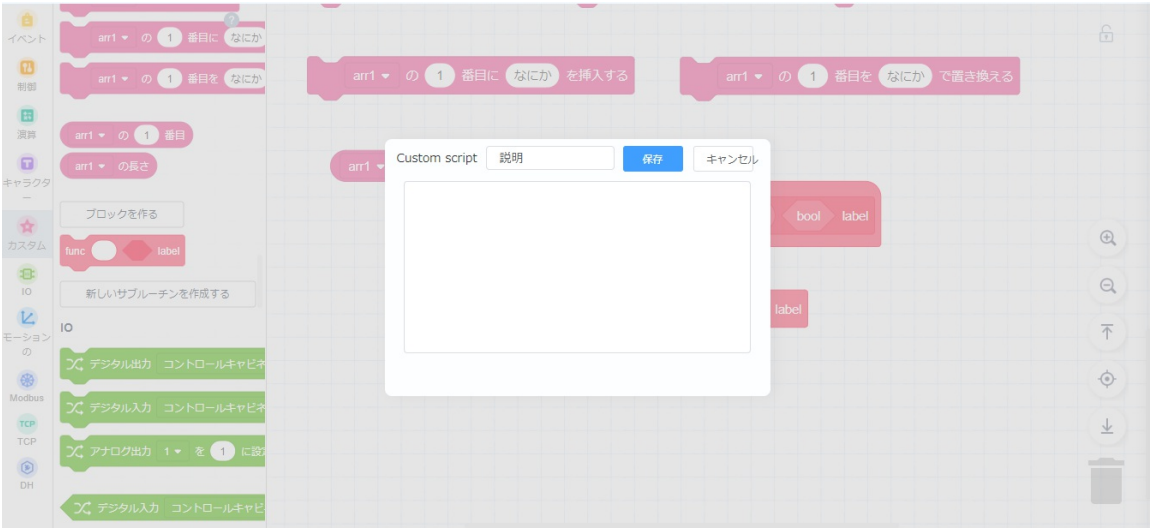
説明：ユーザーがカスタマイズする関数ブロックで、名称と入力パラメータはユーザーにより カスタマイズし、定義済みの関数を呼び出すために使用されます。ブロックリストでブロックを 右クリック（PC端末）/長押し（モバイル端末）すると、この関数の宣言を変更できます。この 関数を削除する必要がある場合、この関数の定義ヘッダーブロックを削除してください。

サブルーチンの新規作成

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

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

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



サブルーチン

- グラフィカルプログラミングのサブルーチン

- スクリプトプログラミングのサブルーチン

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

I/Oブロックグループ

I/Oブロックグループは、ロボットアームのI/O端子の入出力を管理するためのものです。入出力ポートの値範囲は、ロボットアームの対応する端子数により決まります。対応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。

デジタル出力の設定

説明：指定されたDOをオン/オフに設定します。

パラメータ：

- DO端子の位置を選択し、コントロールキャビネットと先端を含みます。
- 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）。

デジタル出力の設定（サブルーチン用）

説明：指定されたDOをオン/オフに設定します。サブルーチンにおいてDOを設定する場合には、このブロックを使用してください。

パラメータ：

- DO端子の位置を選択し、コントロールキャビネットと先端を含みます。
- 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）。

1組のデジタル出力の設定

説明：1組のDOを設定します。ブロックをプログラミングエリアにドラッグした後、クリックして設定します。

パラメータ：

- 「+」または「-」により、設定するDO数量を増減することができます。
- 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）。

デジタル入力の待機

説明：指定したDIが条件を満たすか、タイムアウトしたのを待って、後続のブロックインストラクションを実行します。

パラメータ：

- DI端子の位置を選択し、コントロールキャビネットと先端を含みます。
- DI端子の番号を選択します。
- 待機する状態を選択します（オンまたはオフ）。
- 待機タイムアウト時間。0に設定する場合、条件を満たすまで待ちます。

アナログ出力の設定

説明：指定したアナログ出力値を設定します。

パラメータ：

- アナログ出力端子の番号を選択します。
- 出力する値で、直接記入してもその他戻り値が数値の楕円形のブロックを使用しても構いません。

デジタル入力状態の判断

説明：指定したDI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：

- DI端子の位置を選択し、コントロールキャビネットと先端を含みます。

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

アナログ入力の取得

説明：指定されたアナログ入力値を取得します。

パラメータ：

- アナログ入力端子の位置を選択し、コントロールキャビネットと先端を含みます。
- 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。

移動ブロックグループ

移動ブロックグループは、ロボットアームの移動を制御し、移動関連の設定を行うために使用されます。

移動を制御するブロックは非同期インストラクションです。つまり、インストラクション発行完了後に次のインストラクションを実行し、ロボットの移動完了を待たずに、ロボットは発行された順にインストラクションを1つずつ実行します。既に発行したインストラクションの実行完了を待ってから次のインストラクションを実行する場合、同期インストラクションを使用することができます。

点位置パラメータはこのプロジェクトの「ティーチングポイント」画面で追加した後、ここで選択できます。移動を制御するブロックでは、さらにデフォルトの変数ブロックのドラッグもサポートしています。代わりにその他の戻り値が点位置デカルト座標値の楕円形のブロックを使用します。

高度な構成

事前設定した移動ブロックがプログラミングの要求を満たすことができない場合、高度な構成によりロボットの移動を制御するブロックを作成することができます。作成できたブロックは直接プログラミングエリアに表示されます。詳細は移動の高度な構成を参照してください。

目標点まで移動

説明：ロボットアームが現在の位置から指定された点まで移動することを制御します。ブロックをプログラミングエリアまでドラッグした後、ダブルクリックして高度な構成を行うことができます。詳細は移動の高度な構成を参照してください。

パラメータ：

- 移動方式の選択で、関節移動と直線移動をサポートします。関節移動の軌跡は非線形で、すべての関節は同時に移動を完了します。
- 目標点

目標点まで移動（オフセットあり）

説明：ロボットアームが現在の位置から指定された点のオフセットした後の座標まで移動することを制御し、目標点は現在の点に設定できます。

パラメータ：

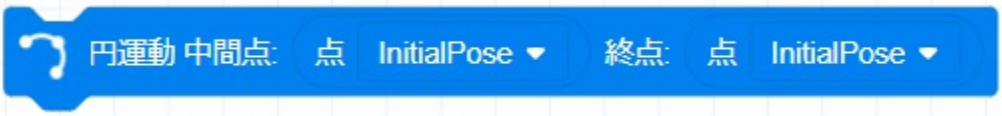
- 移動方式の選択で、相対的な関節の移動と相対的な直線移動をサポートします。
- 目標点
- デカルト座標系で目標点に対応するX軸、Y軸、Z軸方向上のオフセット量。単位：mm。

関節オフセット移動

説明：ロボットアーム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を制御します。

パラメータ：各関節のオフセット量。単位：度。

円弧移動の実行



説明：ロボットアーム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。

パラメータ：

- 中間点とは、円弧を決める中間点のことです。
- 終了点は目標点のことです。

完全円形移動の実行

説明：ロボットアーム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。

パラメータ：

- 中間点は、完全円形を決める中間点のことです。
- 終了点は、完全円形を決める終了点のことです。
- 完全円形移動を行うターン数を入力し、値範囲は1~999です。

軌跡再現

説明：ロボットアームが軌跡再現を行うことを制御します。再現した軌跡のファイルは、軌跡再現プロセス中で記録します。

パラメータ：

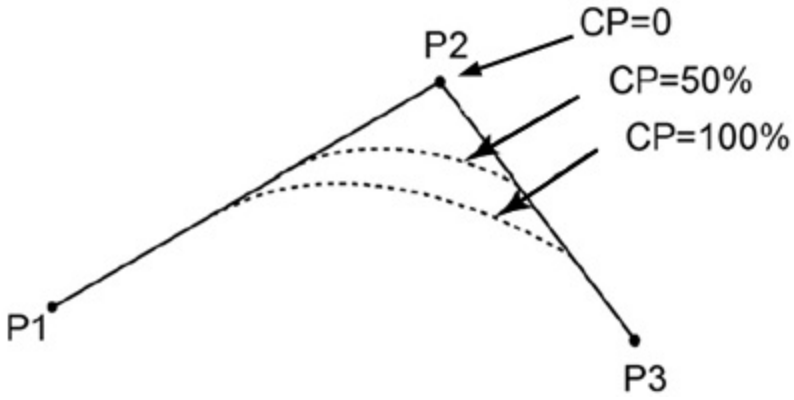
- 再生する軌跡ファイルを選択します。
- 再生時の移動速度を選択します。
  - 等速
  - 0.25倍速
  - 0.5倍速
  - 1倍速
  - 2倍速

同期インストラクション

説明：プログラムがこのインストラクションまで実行すると、ロボットアームが以前に発行したすべてのインストラクションの実行が完了するまで待機し、後のインストラクションを引き続き実行します。

連続経路制御率の設定

説明：移動中の連続経路制御率を設定します。つまり、ロボットアームは開始点から中間点を経て、終点まで到達した場合、下図に示されているように、中間点を経ることは、直角方式で移行するか、曲線方式で移行することです。



パラメータ：連続経路制御率で、値範囲は0~100です。

関節速度比率の設定

説明：関節移動の速度比率を設定します。

パラメータ：関節の速度比率で、値範囲は0~100です。ロボットアームの実際の移動速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

関節加速度比率の設定

説明：関節移動の加速度比率を設定します。

パラメータ：関節の加速度比率で、値範囲は0~100です。ロボットアームの実際の移動加速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

直線速度比率の設定

説明：直線および円弧移動の速度比率を設定します。

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。ロボットアームの実際の移動速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

直線加速度比率の設定

説明：直線および円弧移動の加速度比率を設定します。

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。ロボットアームの実際の移動加速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

指定点の座標値の変更

説明：指定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。
- 変更する座標の次元を選択します。
- 変更後の値を選択します。

指定点の座標値の取得

説明：指定点のデカルト座標の値を取得します。

パラメータ：取得する座標値の点を選択します。

戻り値：指定点のデカルト座標の値を取得します。

指定点の指定座標次元の値の取得

説明：指定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：

- 取得する座標値の点を選択します。
- 取得する座標の次元を選択します。

戻り値：指定点の指定デカルト座標の次元の値をします。

現在位置の指定座標次元の値の取得

説明：TCP現在位置の指定座標系の指定座標次元の値の取得。

パラメータ：

- 取得する座標の次元を選択します。
- ユーザ座標系とツール座標系を選択し、返される座標値は、対応する座標系の値に換算されます。

戻り値：TCP現在位置の指定座標系の指定座標次元の値。

移動の高度な構成

高度な構成方式により、ロボットの移動を制御するブロックを作成します。構成内容は、ブロック名、移動方式、移動パラメータを含みます。それぞれの移動方式では、構成する移動パラメータは異なります。そのうち、ロボットアームの実際の移動速度/移動加速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

MovJ

移動方式：現在の位置から関節補間方式でデカルト座標系の目標位置まで移動します。

基本設定：P点の座標、つまり目標点の座標です。このプロジェクトの「ティーチングポイント」ページで追加した後、ここで選択できます。さらに、直接このページでティーチングしてカスタマイズできます。

高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。
- 過程リモートI/Oの設定：指定距離またはパーセントを移動した場合、指定DOをトリガーします。距離が正である場合、起点からの距離を示します。距離が負である場合、目標点からの距離を示します。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。

MovL

移動方式：現在の位置から、直線補間の方式でデカルト座標系の目標位置まで移動します。

基本設定：P点の座標、つまり目標点の座標です。このプロジェクトの「ティーチングポイント」ページで追加した後、ここで選択できます。さらに、直接このページでティーチングしてカスタマイズできます。

高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。
- 過程リモートI/Oの設定：指定距離またはパーセントを移動した場合、指定DOをトリガーします。距離が正である場合、起点からの距離を示します。距離が負である場合、目標点からの距離を示します。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。

JointMovJ

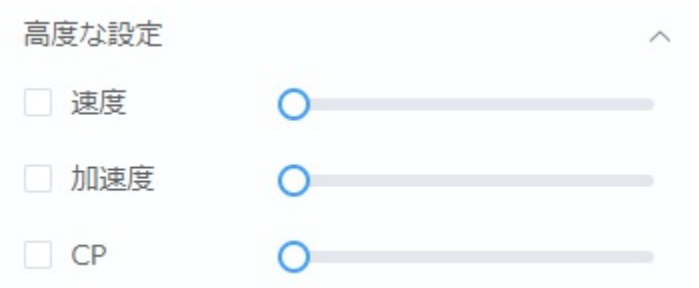
移動方式：現在の位置から関節補間方式で目標の間接角度まで移動します。

基本設定：目標関節角度で、ティーチングによりカスタマイズします。

高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。



RelMovJ

移動方式：現在の位置から関節補間方式でデカルト座標系のオフセット位置まで移動します。

基本設定：デカルト座標系のX軸、Y軸、Z軸方向上のオフセット量。単位：mm。

高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。

RelMovL

移動方式：現在の位置から直線補間方式でデカルト座標系のオフセット位置まで移動します。

基本設定：デカルト座標系のX軸、Y軸、Z軸方向上のオフセット量。単位：mm。

高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。

Arc

移動方式：現在の位置から円弧補間方式でデカルト座標系の指定点まで移動します。現在の位置の座標はAB点で決まる直線上にないようにしてください。

基本設定：

- 中間点Aの座標：円弧の中間点の座標。
- 終点Bの座標：目標点の座標。2つの点はこのプロジェクトの「ティーチングポイント」ページで追加した後、ここで選択するか、直接このページでティーチングしてカスタマイズします。

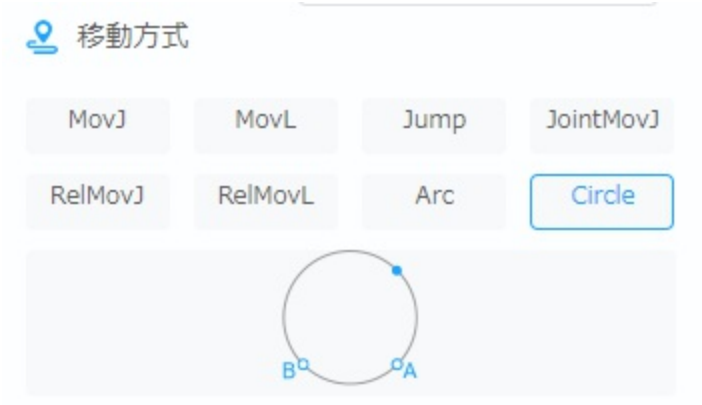
高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。

Circle

移動方式：現在位置から円形補間移動を行い、指定のターン数を移動したら現在位置に戻ります。現在の位置の座標は、AB点で決まる直線上にあってはならず、3つの点で決まる完全な円はロボットアームの移動範囲を超えないでください。



基本設定：

- 中間点Aの座標：完全円形を決める中間点の座標。
- 終点Bの座標：完全円形を決める終点の座標。2つの点はこのプロジェクトの「ティーチングポイント」ページで追加した後、ここで選択するか、直接このページでティーチングしてカスタマイズします。
- サイクル回数：完全円形移動のターン数で、値範囲は1~999です。

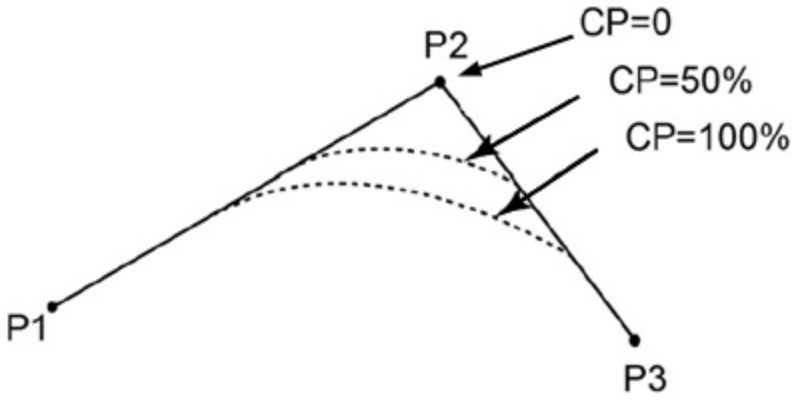
高度な設定：

チェックして、設定する高度なパラメータを構成します。

- 速度(Speed)：移動速度比率で、値範囲は1~100です。
- 加速度(Accel)：移動加速度比率で、値範囲は1~100です。
- 連続経路制御（CP）：移動中の連続経路制御率です。詳細は本章最後の「連続経路制御（CP）の説明」を参照してください。値範囲：0~100。

連続経路制御（**CP**）の説明

連続経路制御とは、ロボットアームが開始点から中間点を経て終点に達する場合、下図に示されているように、中間点を経ることが直角方式で移行か、曲線方式で移行することです。



Modbusブロックグループ

Modbusブロックグループは、Modbus通信関連の操作を行うために用いられます。

Modbusマスターの作成

説明：Modbusのマスターを作成し、スレーブと接続を確立します。

パラメータ：

- ModbusスレーブのIPアドレス。
- Modbusスレーブのポート。
- ModbusスレーブのIDで、値範囲は1~4です。

マスターの作成結果の取得

説明：Modbusマスターの作成結果を取得します。

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4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

入力レジスタの待機

説明：入力レジスタが指定するアドレスの値が条件を満たした後、次のインストラクションを継続して実行します。

パラメータ：

- 入力レジスタの開始アドレスで、値範囲は0~9998です。

- 読み取るデータタイプを選択します。
  - U16：16ビットの符号がない整数（2バイト、1レジスタ占用）
  - U32：32ビットの符号がない整数（4バイト、2レジスタ占用）
  - F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）
  - F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）
- 入力レジスタが指定するアドレスの値が満たす条件。

保持レジスタの待機

説明：保持レジスタが指定するアドレスの値が条件を満たした後、次のインストラクションを継続して実行します。

パラメータ：

- 保持レジスタの開始アドレスで、値範囲は0~9998です。
- 読み取るデータタイプを選択します。
  - U16：16ビットの符号がない整数（2バイト、1レジスタ占用）
  - U32：32ビットの符号がない整数（4バイト、2レジスタ占用）
  - F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）
  - F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）
- 入力レジスタが指定するアドレスの値が満たす条件。

接点レジスタ

説明：接点レジスタが指定するアドレスの値が条件を満たした後、次のインストラクションを継続して実行します。

パラメータ：

- 接点レジスタの開始アドレスで、値範囲は0~9999です。
- 接点レジスタが指定するアドレスの値が満たす条件。

コイルレジスタの待機

説明：コイルレジスタが指定したアドレスの値が条件を満たした後、次のインストラクションを継続して実行します。

パラメータ：

- コイルレジスタの開始アドレスで、値範囲は0~9999です。
- コイルレジスタが指定するアドレスの値が満たす条件。

入力レジスタの読み取り

説明：入力レジスタが指定するアドレスの値を取得します。

パラメータ：

- 入力レジスタの開始アドレスで、値範囲は0~9999です。
- 読み取るデータタイプを選択します。
  - U16：16ビットの符号がない整数（2バイト、1レジスタ占用）
  - U32：32ビットの符号がない整数（4バイト、2レジスタ占用）
  - F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）
  - F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）

戻り値：入力レジスタが指定するアドレスの値。

保持レジスタの読み取り

説明：保持レジスタが指定するアドレスの値を取得します。

パラメータ：

- 保持レジスタの開始アドレスで、値範囲は0~9998です。
- 読み取るデータタイプを選択します。
  - U16：16ビットの符号がない整数（2バイト、1レジスタ占用）
  - U32：32ビットの符号がない整数（4バイト、2レジスタ占用）
  - F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）
  - F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）

戻り値：入力レジスタが指定するアドレスの値。

接点レジスタの読み取り

説明：接点レジスタが指定するアドレスの値を取得します。

パラメータ：接点レジスタの開始アドレスで、値範囲は0~9998です。

戻り値：接点レジスタが指定するアドレスの値。

コイルレジスタの読み取り

説明：コイルレジスタが指定するアドレスの値を取得します。

パラメータ：コイルレジスタの開始アドレスで、値範囲は0~9999です。

戻り値：コイルレジスタが指定するアドレスの値。

コイルレジスタの連続読み取り

説明：コイルレジスタが指定するアドレスの値を連続で読み取ります。

パラメータ：

- コイルレジスタの開始アドレスで、値範囲は0~9999です。
- 連続で読み取ったレジスタバイト数。端末経由でスレーブと通信する場合の最大値は216（CR）または984（Nova）、それ以外の場合の最大値は2008です。

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

保持レジスタの連続読み取り

説明：保持レジスタが指定するアドレスの値を連続で読み取ります。

パラメータ：

- 保持レジスタの開始アドレスで、値範囲は0~9998です。
- 連続で読み取った値の数。
- 読み取るデータタイプを選択します。

- U16：16ビットの符号がない整数（2バイト、1レジスタ占用）。端末経由でスレーブと通信する場合は最大13個（CR）または61個（Nova）、その他の場合は最大125個の値を連続して読み取ることができます。
- U32：32ビットの符号がない整数（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）。端末経由でスレーブと通信する場合は最大3個（CR）または15個（Nova）、その他の場合は最大31個の値を連続して読み取ることができます。

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

コイルレジスタへの書き込み

説明：指定する値をコイルレジスタが指定するアドレスに書き込みます。

パラメータ：

- コイルレジスタの開始アドレスで、値範囲は6~9999です。
- 選択して書き込む値で、0または1のみです。

コイルレジスタへの連続書き込み

説明：指定する値をコイルレジスタが指定するアドレスに連続して書き込みます。

パラメータ：

- コイルレジスタの開始アドレスで、値範囲は0~9999です。
- 書き込む値を入力し、配列を埋めます。各ビットは0または1のみです。端末経由でスレーブと通信する場合、一度に最大440個の値、その他の場合は一度に最大1976個の値を書き込むことができます。

保持レジスタへの書き込み

説明：指定する値を保持レジスタが指定するアドレスに書き込みます。

パラメータ：

- 保持レジスタの開始アドレスで、値範囲は0~9998です。
- 書き込む値を選択し、選択したデータタイプに対応していなければなりません。
- 書き込むデータタイプを選択します。
  - U16：16ビットの符号がない整数（2バイト、1レジスタ占用）。端末経由でスレーブと通信する場合は最大27個、その他の場合は最大123個の値を連続して書き込むことができます。
  - U32：32ビットの符号がない整数（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大13個、その他の場合は最大61個の値を連続して書き込むことができます。
  - F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大13個、その他の場合は最大61個の値を連続して書き込むことができます。
  - F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）。端末経由でスレーブと通信する場合は最大6個、その他の場合は最大30個の値を連続して書き込むことができます。

マスタークローズ

説明：Modbusマスターを閉じて、すべてのスレーブとの接続を解除します。

TCPブロックグループ

TCPブロックグループはTCP関連の操作を行うために用いられます。

SOCKETの接続

詳細：TCPクライアントを確立し、指定したTCPサーバーと通信を行います。

パラメータ：

- SOCKETのシリアル番号を選択し、最大で4つのTCP通信リンクを確立します。
- TCPサーバーのIPアドレス。
- TCPサーバーのポート。

SOCKET接続結果の取得

詳細：TCP通信接続の結果を取得します。

パラメータ：SOCKETシリアル番号を選択します。

戻り値：接続に成功すると0を戻し、接続に失敗すると1を戻します。

SOCKETの確立

詳細：TCPサーバーを確立し、クライアントによる接続を待機します。

パラメータ：

- SOCKETのシリアル番号を選択し、最大で4つのTCP通信リンクを確立します。
- TCPサーバーのIPアドレス。
- TCPサーバーのポート。502または8080に設定してはなりませんのでご注意ください。さもなければModbusのデフォルトポートまたはパイプラインの動的追跡に使用されるポートと競合し、TCPサーバー確立の失敗をもたらす可能性があります。

SOCKET確立結果の取得

詳細：TCPサーバー確立の結果を取得します。

パラメータ：SOCKETシリアル番号を選択します。

戻り値：作成が完了した場合の戻り値は0で、作成が失敗した場合の戻り値は1です。

SOCKETの閉鎖

詳細：指定したSOCKETを閉じて、当該通信リンクを切断します。

パラメータ：SOCKETシリアル番号を選択します。

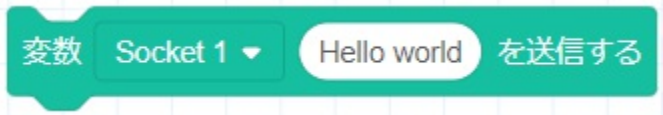
変数の取得

詳細：TCP通信を介して変数を取得し保存します。

パラメータ：

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

変数の送信



詳細：TCP通信を介して変数を送信します。

パラメータ：

- SOCKETシリアル番号を選択します。

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

変数送信結果の取得

詳細：TCP変数送信の結果を取得します。

パラメータ：SOCKETシリアル番号を選択します。

戻り値：送信に成功すると0を戻し、送信に失敗すると1を戻します。

付属書C スクリプトプログラミング関数の説明

Lua基本文法

変数とデータのタイプ

> Luaプログラミングの関連知識を系統的に学習したい場合、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。本マニュアルでは、迅速に調べられるようにするため、一部のLua基本文法を取り上げます。

変数は値を保存するためのもので、値をパラメータとして渡すか、結果を返します。変数は「=」により割り当てられます。

Localを使用してローカル変数として明示的に宣言していない限り、Luaの変数はデフォルトでグローバル変数です。ローカル変数のアクションスコープは、宣言した位置からステートメントブロックまでです。

```
a = 5               -- グローバル変数
local b = 5         -- ローカル変数
```

変数名は、Luaで予約されているキーワードを除き、数字で始まらないアルファベット、アンダーライン、数字の文字列にすることができます。

Lua変数はタイプ定義が不要で、変数のために割り当てられるだけです。Luaは値に応じて変数のタイプを自動的に判断します。

Luaはさまざまなデータタイプをサポートします。比較的一般的であるのは数字（number）、ブール値（boolean）、文字列（string）、テーブル（table）です。Luaのアレーはtableの一種です。

Luaには、さらに特殊なデータタイプであるnilがあります。nilは空（有効値が何もない）を示します。例えば、割り当てられていない変数をプリントすると、1つのnil値を出力します。

数字

Luaのnumberは倍精度タイプの実数浮動小数点であり、さまざまな演算に対応しています。以下の書き方はすべてnumberとみなされています。

- 2
- 2.2
- 0.2
- 2e+1
- 0.2e-1
- 7.8263692594256e-06

ブール値

booleanタイプは2つの値しかありません：trueとfalse。Luaはfalseとnilをfalseとしています。その他はtrueです。数字の0もtrueです。

文字列

stringは数字、文字、アンダーラインで構成される文字列です。stringは以下の3種類の方法で示すことができます。

- 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**]]**間の文字列。

1つの数字文字列で算術操作を行う場合、Luaはこの数字文字列を1つの数字にしようとします。

Luaは多くの方法を提供し、文字列の操作をサポートしています。

方法	説明
string.upper (argument)	文字列をすべて大文字に変換します
string.lower (argument)	文字列をすべて小文字に変換します
string.gsub(mainString, findString,replaceString,num)	文字列中で置き換えます。mainStringは操作する文字列で、findStringは置き換えられる文字で、replaceStringは置き換える文字で、numは置換回数です（省略可能、つまりすべて置換）
string.find (str, substr, [init, [end]])	指定された目標文字列strにおいて指定された内容substrを検索します。マッチングする部分文字列を見つけた場合、この部分文字列の開始インデックスと終了インデックスを返します。存在しない場合にはnilを返します。
string.reverse(arg)	文字列の反転
string.format(...)	printfに類似した形式の文字列を返します
string.char(arg) 和 string.byte(arg[,int])	charは整数を文字列に変換して接続します。byteは文字列を整数に変換します（特定の文字を指定できます。デフォルトは1番目の文字です）
string.len(arg)	文字列長さを計算します
string.rep(string, n)	文字列stringのn個のコピーを返します
..	2つの文字列の連結
string.gmatch(str, pattern)	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、1つの文字列str内で見つけた次のpatternの説明に一致する部分文字列を返します。パラメータpatterの説明の文字列が見つからない場合、イテレータ関数はnilを返します
string.match(str, pattern, init)	String match()はソース文字列str中の最初ペアのみを検索します。パラメータinitはオプションで、検索プロセスの起点を指定します。デフォルトは1です。 ペアリングが成功すると、関数はペアリング表現式中のすべてのキャプチャ結果を返します。キャプチャフラグが設定されていない場合、ペアリング文字列全体を返します。ペアリングが成功しなかった場合、nilを返します
string.sub(s, i [, j])	文字列を切り取るために使用されます。sは切り取る文字列で、iは切り取りの開始位置、jは切り取りの終了位置で

	す。デフォルトは-1で、最後の1文字です。

例：

```
str = "Lua"
print(string.upper(str))        --すべての文字列を大文字に変換し、結果をプリント：LUA
print(string.lower(str))        --すべての文字列を小文字に変換し、結果をプリント：lua
print(string.reverse(str))    --文字列を反転し、結果をプリント：aul
print(string.len("abc"))        --文字列abcの長さを計算し、結果をプリント：3
print(string.format("the value is: %d",4))     --結果をプリント：the value is: 4
print(string.rep(str,2))        --文字列を2回コピーし、結果をプリント：LuaLua
  string1 = "cn."
  string2 = "dobot"
  string3 = ".cc"
  print("文字列を接続する",string1..string2..string3)  -- ..を使用して文字列を接続し、結果をプリント：cn.dobot.cc

  string1 = [[aaaa]]
  print(string.gsub(string1,"a","z",3))               --文字列中で置き換え、結果をプリント：zzza

  print(string.find("Hello Lua user", "Lua", 1))        --文字列中でLuaを検索し、部分文字列の開始インデックスと終了インデックスを返し、結果をプリント：7, 9

  sourcestr = "prefix--runoobgoogletaobao--suffix"
  sub = string.sub(sourcestr, 1, 8)                   --1番目から8番目までの文字列プレフィックスを取る
  print("\n切り取り", string.format("%q", sub))            --結果をプリント："prefix--"を切り取る
```

テーブル

tableは索引があるデータです。

- 最も簡単な構造関数は{}で、空のテーブルを作成します。テーブルを直接初期化できます。
- tableは実際には関連型アレーで、任意のタイプの値を用いてアレーのインデックスにすることができます。ただし、この値はnilであってはなりません。
- tableは大きさが固定されたものではありません。自身の必要性に応じて拡大することができます。
- 符号によりテーブル長さを取得できます。

  ```
  tbl = {[1] = 2, [2] = 6, [3] = 34, [4] =5}
  print("tbl長さ ", #tbl)  -- 結果プリントは4
  ```

Luaは多くの方法によりテーブルの操作をサポートしています。

方法	説明
table.concat (table [, sep	concatはconcatenaet（連鎖、結合）の短縮形です。table.concat()関数はパラメータでtableを指定したアレー部分のstart位置からend位置までのす

[, start [, end]]])	べての要素を列記します。要素間は指定された分割符号(sep)で分けられます
table.insert (table, [pos,] value)	tableのアレー部分の指定された位置(sep)に値がvalueである1つの要素を挿入します。posパラメータはオプションで、デフォルトはテーブルの末尾です
table.remove (table [, pos])	tableアレー部分がposの位置にある要素を返します。その後の要素は前に移動します。posパラメータはオプションで、デフォルトはtable長さです。つまり、最後の1つの要素から削除します
table.sort (table [, comp])	指定されたtableを昇順に並び替えます

例1：

```
fruits = {}         -- テーブルを初期化する
fruits = {"banana","orange","apple"}  -- テーブルに割り当てる

print("接続後の文字列 ",table.concat(fruits,", ", 2,3))   -- インデックスを指定してtableを接続し、接続後の文字列はorange, apple

-- 最後に挿入する
table.insert(fruits,"mango")
print("インデックスが4の要素は ",fruits[4])      --結果をプリント：インデックスが4の要素はmango

-- インデックスが2のキーに挿入する
table.insert(fruits,2,"grapes")
print("インデックス2の要素は ",fruits[2])      --結果をプリント：インデックスが2の要素はgrapes

print("最後の要素は ",fruits[5])         --結果をプリント：最後の要素はmango
table.remove(fruits)
print("削除後の最後の要素は ",fruits[5])   --結果をプリント：削除後の最後の要素はnil
```

例2：

```
fruits = {"banana","orange","apple","grapes"}
print("ソート前")
for k,v in ipairs(fruits)
do
        print(k,v)          --結果をプリント：banana orange apple grapes
end
--昇順にソート
table.sort(fruits)
print("ソート後")
for k,v in ipairs(fruits) do
        print(k,v)          --結果をプリント：apple banana grapes orange
end
```

アレー

アレーは、同じデータタイプの要素を一定の順序で並べた集合です。1次元アレーや多次元アレーにすることができます。Luaアレーのインデックスキー値は、整数で表すことができます。アレーの大きさは固定されたものではありません。

- 1次元アレー：最も簡単なアレーで、その論理構造は線形リストです。
- 多次元アレー：つまり、アレーにアレーを含む、または1次元アレーのインデックスキーが1つのアレーに対応します。

例1：1次元アレーはforを用いてアレー中の要素を循環できます。整数インデックスを使用して、アレー要素にアクセスします。インデックスに値がない場合にはnilを返します。

```
array = {"Lua", "Tutorial"}  --1次元アレーを作成する
for i= 0, 2 do
   print(array[i])           --結果プリントはそれぞれ：nil Lua Tutorial
end
```

Luaインデックス値は1から開始しますが、0を指定して開始することもできます。これ以外にも、負の数をアレーのインデックス値とすることもできます。

```
array = {}
for i= -2, 2 do
   array[i] = i*2+1           --1次元アレーに割り当てる
end
for i = -2,2 do
   print(array[i])           --結果プリントはそれぞれ：-3 -1 1 3 5
end
```

例2：3 x 3行列の多次元アレー

```
-- アレーを初期化する
array = {}
for i=1,3 do
 array[i] = {}
    for j=1,3 do
      array[i][j] = i*j
    end
end

-- アレーにアクセスする
for i=1,3 do
  for j=1,3 do
     print(array[i][j])     --結果プリントはそれぞれ：1 2 3 2 4 6 3 6 9
  end
end
```

演算子

算術演算子

インストラクション記号	説明
+	加算
-	減算
*	乗算
/	浮動小数点除算
//	切り捨て除算
%	切り上げ除算
^	指数演算
&	ビットAND演算
\	ビットOR演算
~	ビットXOR演算
<<	ビット左シフト演算
>>	ビット右シフト演算

例：

```
a=20
b=5
print(a+b)            --a+bの結果をプリント：25
print(a-b)            --a-bの結果をプリント：15
print(a*b)            --axbの結果をプリント：100
print(a/b)            --a/bの結果をプリント：4
print(a//b)            --aがbで割り切れる結果をプリント：4
print(a%b)            --aをbで割った余りの結果をプリント：0
print(a^b)            --aのb乗の結果をプリント：3200000
print(a&b)            --aとbのビットANDの結果をプリント：4
print(a|b)            --aとbのビットORの結果をプリント：21
print(a~b)            --aとbのビットXORの結果をプリント：17
print(a<<b)            --aをb単位で左に移動した結果をプリント：640
print(a>>b)            --aをb単位で右に移動した結果をプリント：0
```

関係演算子

インストラクション記号	説明
==	等しい

~=	等しくない
<=	以下
>=	以上
<	未満
>	より大きい

例：

```
a=20                --変数aを作成する
b=5                   --変数bを作成する
print(a==b)           --aがbに等しい比較結果をプリント：false
print(a~=b)           --aがbに等しくない比較結果をプリント：true
print(a<=b)           --aがb以下である比較結果をプリント：false
print(a>=b)           --aがb以上である比較結果をプリント：true
print(a<b)          --aがb未満である比較結果をプリント：false
print(a>b)          --aがbより大きい比較結果をプリント：true
```

ロジック演算子

インストラクション記号	説明
and	論理積、両側がtrueである場合、その結果はtrue。片側がfalseである場合、その結果はfalse
or	論理和、片側の結果がtrueである場合、その結果はtrue。orの両側がfalseである場合、その結果はfalse
not	論理否定とは、判断結果を直接逆にする

```
a=true
b=false
print(a and b)            --trueとfalse、結果はfalse
print(a or b)           --trueとfalse、結果はtrue
print(20 > 5 not true)    --trueとuntrueは、trueとfalseに等しい、最後の結果はfalse
```

フロー制御

インストラクション記号	説明
if…then…elseif…then…else…end	ifは条件判断インストラクション。上から下に順次条件が成立しているかどうかを判断します。特定の判断がtrueである場合、対応するコードブロックの実行を完了し、後の条件判断を直接無視し、実行しません。
while…do…end	while循環制御インストラクション。条件がtrueである場合、プログラムは特定のステートメントを繰り返し実行します。ステートメントを実行する前に、条件がtrueであるかどうかをチェックします。
for…do…end	for循環制御インストラクション。指定されたステートメントを繰り返し実行します。繰り返し回数はforステートメントで制御されます
repeat…until()	repeat循環制御インストラクション。指定された条件がtrueになるまで、サイクルを繰り返し実行します。

例：

1 if条件判断インストラクション

```
a = 100;
b = 200;
--[ 検査条件 --]
if(a == 100)
then
   --[if条件がtrueである場合、以下のif条件判断を実行する--]
   if(b == 200)
   then
      --[if条件がtrueである場合、このステートメントブロックを実行する--]
     print("aの値: ", a );
  print("bの値: ", b );
   end
end
```

1. while循環制御インストラクション

```
a=10
while( a < 20 )
do
   print("aの値: ", a)
   a = a+1
end
```

1. for循環制御インストラクション

```
for i=10,1,-1 do
    print(i)
end
```

1. repeat循環制御インストラクション

```
a = 10
repeat
   print("aの値: ", a)
   a = a + 1
until(a > 15)
```

関数の説明

全般的な説明

移動方法

ロボットアームがサポートする移動方法は次のいくつかに分けられます。

関節移動

ロボットアームは現在の各関節角度と目標点の各関節角度の差に基づいて各関節の移動を計画し、すべての関節が同時に移動を完了します。関節移動はTCP（Tool Center Point）の移動の軌跡を制約せず、通常この軌跡は直線ではありません。

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

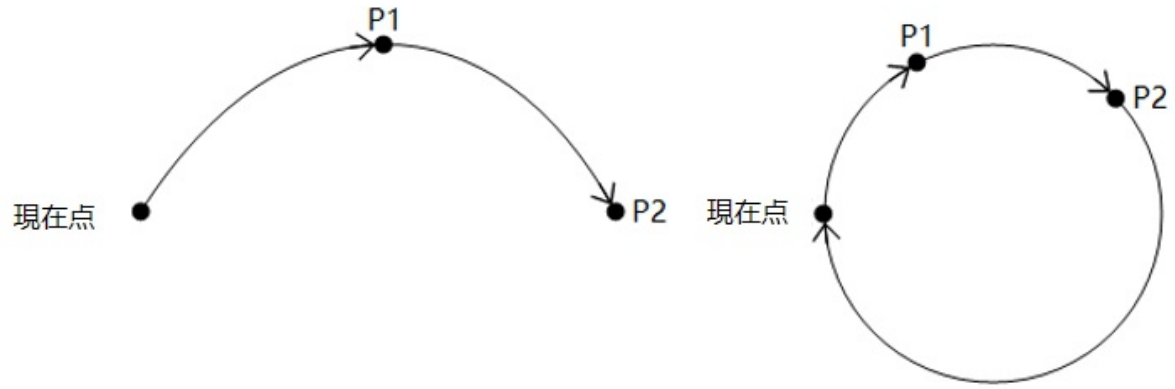
直線移動

ロボットアームは現在のポーズと目標点のポーズに基づいて移動の軌跡を計画し、TCPの移動の軌跡が直線になり、先端の姿勢が移動の過程で一定の速度で変化します。

移動の軌跡が特異位置を通る場合、直線移動の命令をロボットアームに送信するとエラーが発生する可能性があります。その場合は位置を再計画するか、特異位置近くで関節移動を使用することをお勧めします。

弧線移動

ロボットアームは現在の位置、P1、P2の3つの非共線点を使用して円弧または完全な円を確定します。移動中のロボットアームの先端の姿勢は、現在の点とP2点の姿勢から補間計算され、P1点の姿勢は計算に参加しません（つまり、ロボットアームがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。



移動の軌跡が特異位置を通る場合、弧線移動の命令をロボットアームに送信するとエラーが発生する可能性があります。その場合は位置を再計画するか、特異位置近くで関節移動を使用することをお勧めします。

点パラメータ

命令内の点パラメータは、通常、点位置保存リスト内の教示点を使用し、教示点の名前を使用して直接呼び出すことができます。実際に教示点は以下の形式で定数としてバックグラウンドに保存されます。

```
--[[
    name：教示点の名称。
    joint：教示点の関節座標。
    tool：教示時に使用する工具座標系のインデックス。
    user：教示時に使用するユーザー座標系のインデックス。
    pose：教示点のデカルト座標。
--]]
{
    name = "name",
    joint = {j1, j2, j3, j4, j5, j6},
    tool = index,
    user = index,
    pose = {x, y, z, rx, ry, rz}
}
```

点パラメータとして教示点以外にもデカルト座標点や関節座標点をカスタマイズすることも可能です（詳細は各命令のパラメータ説明を参照）。

デカルト座標点：

```
{pose = {x, y, z, rx, ry, rz} }
```

関節座標点：

```
{joint = {j1, j2, j3, j4, j5, j6} }
```

速度パラメータ

オプションパラメータのSpeed/SpeedSとAccel/AccelSは、ロボットアームが移動命令を実行する際の速度と加速度の比率を指定します。

```
ロボットアームの実移動速度/加速度＝最高速度/加速度×グローバル速度×命令速度
```

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

グローバル速度は、制御ソフトウェア（上図右上隅）またはSpeedFactor命令で設定できます。

命令の速度は、移動命令のオプションパラメータで指定された比率で、移動加速度/速度比率がオプションパラメータで指定されていない場合、デフォルトでは移動パラメータで設定された値が使用されます（詳細はSpeed、Accel、SpeedS、AccelS、SpeedR、AccelR命令を参照、命令を呼び出して設定しない場合のデフォルト値はすべて50）。

例：

```
Accel(50) -- 関節移動のデフォルト加速度を50%に設定
Speed(60) -- 関節移動のデフォルト速度を60%に設定
AccelS(70) -- 直線移動のデフォルト加速度を70%に設定
SpeedS(80) -- 直線移動のデフォルト速度を80%に設定

-- 全体速度比率は20%;

Go(P1) -- （最大関節加速度 x 20%x 50%）の加速度と（最大関節速度 x 20% x 60%）の速度で関節をP1に移動
Go(P2,"Speed = 40, Accel = 90") -- （最大関節加速度 20%x 30%）の加速度と（最大関節速度 x 20% x 80%）の速度で関節をP1に移動
```

```
Move(P1) -- （最大デカルト加速度 x 20% x 70%）の加速度と（最大デカルト速度 x 20% x 80%）の速度でP1に直線移動
Move(P1,"SpeedS = 40, AccelS = 90")  -- （最大デカルト加速度 x 20% x 40%）の加速度と（最大デカルト速度 x 20% x 90%）の速度でP1に直線移動
```

即時命令とキュー命令

Dobotが提供する命令は、即時命令とキュー命令に分かれています。

- 即時命令は発行後すぐに実行されます。
- キュー命令は発行後、すぐには実行されず、バックグラウンド アルゴリズム キューに入り、実行を待ちます。

バックグラウンド アルゴリズム キューはスレッドをブロックしません。キュー命令の後に即時命令が呼び出された場合、次の例のように、キュー命令が完了する前に即時命令が実行される可能性があります。

```
Go(P1)  -- キュー命令
local currentPose = GetPose()  -- 即時命令
```

この例では、GetPose()で取得した点はP1ではなく、動作中のプロセス点となります。

即時命令の実行時に、前のすべての命令が実行されていることを確保したい場合は、即時命令を呼び出す前にSync()命令を呼び出すことができます。この命令は、次のように、前のすべての命令が実行されるまでプログラムの実行をブロックします。

```
Go(P1)  -- キュー命令
Sync()
local currentPose = GetPose()  -- 即時命令
```

この例では、GetPose()で取得した点はP1です。

移動インストラクション

移動インストラクション関数はロボットアームの移動を制御するために使用されます。このグループの命令はすべてキュー命令です。

Go

プロトタイプ：

```
Go(P,"User=1 Tool=2 CP=1 Speed=50 Accel=20 SYNC=1")
```

説明：

現在位置から、デカルト座標系における目標位置までポイントツーポイント（関節移動）方式で移動します。関節移動の軌跡は非線形で、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：

P：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
Go(P1)
```

ロボットアームは、デフォルト設定に従ってポイントツーポイント方式で点P1に移動します。

Move

プロトタイプ：

```
Move(P,"User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

説明：

現在位置から、デカルト座標系における目標位置に直線方式で移動します。

必須パラメータ：

P：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。
- STOP：動作停止入力。DI端子の番号を指定し、この動作命令実行中に指定したDIがONになると、ロボットは直ちに動作を停止し、次の命令を直接実行します。このパラメータが設定されている場合、SYNCパラメータの設定は無効になり、デフォルトは1になります。

例：

```
Move(P1)
```

ロボットアームは、デフォルト設定に従って点P1に直線移動します。

```
Move(P1,"STOP=3")
Move(P2)
```

ロボットアームは点P1に向かって直線移動しますが、移動中にDI3がONになると直ちにP1に向かう移動を停止し、P2点に向かって直線移動します。

MoveJ

プロトタイプ：

```
MoveJ(P,"CP=1 Speed=50 Accel=20 SYNC=1")
```

説明：

現在位置から、ポイントツーポイント（関節移動）方式で目標関節角度に移動します。

必須パラメータ：

P：目標点を表します。関節角度のみで定義することができます。

オプションパラメータ：

- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
local P = {joint={0,-0.0674194,0,0,0,0}}
MoveJ(P)
```

関節座標点Pをカスタマイズします。ロボットアームは、デフォルト設定に従って点Pに移動します。

Circle3

プロトタイプ：

```
Circle3(P1,P2,Count,"User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

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。現在位置、P1、P2の3つの点で1つの完全な円形を画定する必要があるため、現在位置はP1とP2で決められる直線上にあってはならず、かつ3つの点で決められる完全な円形はロボットアームの範囲外にあってはなりません。

必須パラメータ：

- P1：完全な円形の中間点を表します。該プロジェクトの「ティーチング点」ページで追加してからここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標点位置のみに対応します。
- P2：完全な円形の終了点を表します。該プロジェクトの「ティーチング点」ページで追加

してからここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標点位置のみに対応します。
- Count：完全円形移動を行う回数ををを表します。値範囲は1~999とします。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
Go(P1)
Circle3(P2,P3,1)
```

ロボットアームはP1に移動してから、P1、P2、P3で決められる完全な円形に沿って一回り移動します。

Arc3

プロトタイプ：

```
Arc3(P1,P2,"User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

説明：

現在位置から、円弧補完方式でデカルト座標系における指定した点に移動します。現在位置、P1、P2の3つの点で1つの円弧を決める必要があるため、現在位置はP1とP2で決められる直線上にあってはなりません。

必須パラメータ：

- P1：円弧の中間点を表します。該プロジェクトの「ティーチング点」ページで追加してからここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標点位置のみに対応します。
- P2：目標点を表します。該プロジェクトの「ティーチング点」ページで追加してからここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標点位置のみに対応します。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
Go(P1)
Arc3(P2,P3)
```

ロボットアームはP1に移動してから、P2を経由してP3に円弧移動します。

GoIO

プロトタイプ：

```
GoIO(P,{ {Mode,Distance,Index,Status},{Mode,Distance,Index,Status}...}, "User=1 Tool=2 CP=1 Sp
eed=50 Accel=20 SYNC=1")
```

説明：

現在位置から、デカルト座標系における目標位置までポイントツーポイント（関節移動）方式で移動します。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：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。
- 並行デジタル出力パラメータ：ロボットアームが指定した距離または百分率に移動すると、指定したDOをトリガーするように設定します。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は距離百分率を表し、1は距離値を表します。
  - Distance：指定した距離。
    - Distanceが正数であるとき、開始点までの距離を表します。
    - Distanceが負数であるとき、目標点までの距離を表します。
    - Modeが0であるとき、Distanceは総距離に対する百分率を表します。値範囲は

1~100とします。
        - Modeが1であるとき、Distanceは距離の値を表します。単位：mm。
    - Index：DO端子の番号。
    - Status：設置対象のDO状態。0はオフを表し、1はオンを表します。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
GoIO(P1, {0, 10, 2, 1})
```

ロボットアームはデフォルト設定に従って点P1に移動し、開始点まで10%の位置に移動すると、DO2がオンになるように設定します。

MoveIO

プロトタイプ：

```
MoveIO(P,{ {Mode,Distance,Index,Status},{Mode,Distance,Index,Status}},"User=1 Tool=2 CP=1 Spee
dS=50 AccelS=20 SYNC=1")
```

説明：

現在位置から、デカルト座標系における目標位置に直線方式で移動します。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：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。
- 並行デジタル出力パラメータ：ロボットアームが指定した距離または百分率に移動すると、指定したDOをトリガーするように設定します。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は距離百分率を表し、1は距離値を表します。
    - Distance：指定した距離。
        - Distanceが正数であるとき、開始点までの距離を表します。
        - Distanceが負数であるとき、目標点までの距離を表します。
        - Modeが0であるとき、Distanceは総距離に対する百分率を表します。値範囲は1~100とします。
        - Modeが1であるとき、Distanceは距離の値を表します。単位：mm。
    - Index：DO端子の番号。
    - Status：設置対象のDO状態。0はオフを表し、1はオンを表します。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
MoveIO(P1, {0, 10, 2, 1})
```

ロボットアームはデフォルト設定に従って点P1に移動し、開始点まで10%の位置に移動すると、DO2がオンになるように設定します。

MoveJIO

プロトタイプ：

```
MoveJIO(P,{ {Mode,Distance,Index,Status},{Mode,Distance,Index,Status}...}, "User=1 Tool=2 CP=1
 Speed=50 Accel=20 SYNC=1")
```

説明：

現在の位置から、ポイントツーポイント（関節移動）方式により目標関節角度まで移動します。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：目標関節角度を表します。

- 並行デジタル出力パラメータ：ロボットアームが指定距離または百分率まで移動する場合に、指定のDOを起動することを設定します。複数グループを設定することができます。各グループは以下のパラメータを含みます。
  - Mode：トリガモード。0は距離百分率を表し、1は距離値を表します。
  - Distance：指定した距離。
    - Distanceが正数であるとき、開始点までの距離を表します。
    - Distanceが負数であるとき、目標点までの距離を表します。
    - Modeが0であるとき、Distanceは総距離に対する百分率を表します。値の範囲は1～100とします。
    - Modeが1であるとき、Distanceは距離の値を表します。単位：mm。
  - Index：DO端子の番号。
  - Status：設置対象のDO状態。0はオフを表し、1はオンを表します。

オプションのパラメータ：

- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCP命令を参照）。値の範囲：0～100。
- Speed：移動速度比率。値の範囲：1～100。
- Accel：移動加速度比率。値の範囲：1～100。
- SYNC：同期フラグ、値の範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出し後は命令が完全に実行されるまで戻りません。

例：

```
local P = {joint={0,-0.0674194,0,0,0,0}}
MoveJIO(P, { {0, 10, 2, 1} })
```

ロボットアームはデフォルト設定に従ってP点に移動し、開始点まで10%の位置に移動すると、DO2がオンになるように設定します。

ServoJ

プロトタイプ

```
ServoJ(P,"t=0.1 lookahead_time=50 gain=500")
```

説明

関節空間に基づく動的追従命令は通常、オンライン制御のインチング機能に使用され、動的追従はループで呼び出されることによって実現されます。呼び出しの頻度は33Hz、つまり呼び出しのループ間隔は30msに設定することをお勧めします。

⚠️ 注意：

- このコマンドはグローバル速度の影響を受けません。
- 目標点と現在点の関節角度差が大きく、移動時間tが小さい場合、関節の移動速度が速すぎて、サーボがエラーを報告したり、本体が電源異常になりシャットダウンする可能性があります。
- この命令を呼び出す前に、ロボットが目標点をスムーズに追跡できるように、動作点の速度を計画し、一定の時間間隔tで速度計画済みの点を発行することをお勧めします。

必須パラメータ

P：目標関節角度を表します。

オプションのパラメータ

- t：この点の実行時間（s）。値の範囲は[0.02,3600.0]で、デフォルト値は0.1です。
- lookahead_time：PID制御の項目Dと同様な働きをする予備量。基準量（単位なし）。値の範囲は[20.0,100.0]、デフォルト値は50です。
- gain：PID制御の項目Pと同様な働きをする目標位置の比例ゲイン。基準量（単位なし）。値の範囲は[200.0、1000.0]、デフォルト値は500です。

パラメータlookahead_timeとgainは、ロボットアームの動きの応答時間と軌道の滑らかさを共同で決定します。小さいlookahead_time値または大きいgain値により、ロボットアームの応答は速くなりますが、不安定さやジッターが発生するおそれがあります。

例

```
ServoJ({joint={0,-0.0674194,0,0,0,0}},"t=0.1 lookahead_time=50 gain=500")
```

ServoP

プロトタイプ

```
ServoP(P)
```

説明

デカルト空間に基づく動的追従命令は通常、オンライン制御のインチング機能に使用され、動的追従はループで呼び出されることによって実現されます。呼び出しの頻度は33Hz、つまり呼び出しのループ間隔は30msに設定することをお勧めします。

パラメータ

P：目標のデカルト点を示します。

例

```
ServoP({pose={-500,100,200,150,0,90} }）
```

移動パラメータ

移動パラメータ関数は、ロボットアーム移動関連パラメータを設定または取得するために使用されます。GetPoseとGetAngleを除いて、このグループの命令はすべてキュー命令です。

Sync

プロトタイプ：

```
Sync()
```

説明：

プログラム実行キューインストラクションをブロックし、すべてのキューインストラクションの実行が完了した後に戻り、後続のインストラクションを実行します。通常、ロボットアーム移動の完了を待機するために使用されます。

例：

```
Go(P1)
Go(P2)
Sync()
```

ロボットアームがまずP1に移動してから、P2に移動するのを待機してから戻り、後続のインストラクションを実行します。

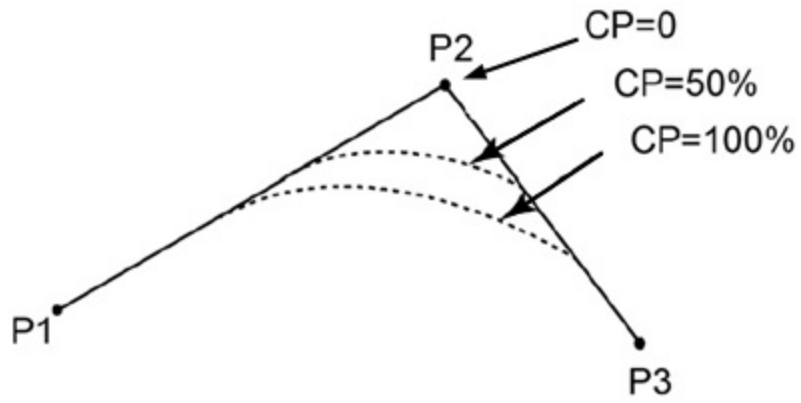
CP

プロトタイプ：

```
CP(R)
```

説明：

連続経路制御の百分率を設定します。すなわち、ロボットアームが複数の点を経由して連続的に移動するとき、直角方式でそれとも曲線方式で中間点を通過するのかを設定します。



必須パラメータ：

R：連続経路制御の百分率。値範囲：0~100。

例：

```
CP(50)
Move(P1)
Move(P2)
Move(P3)
```

P1から点P3に移動するロボットアームは、点P2を通過するとき50%で連続経路制御を行います。

SpeedFactor

プロトタイプ：

```
SpeedFactor(R)
```

説明：

グローバル速度比率を設定します。

必須パラメータ：

R：速度比率。値の範囲：1～100。

例：

```
SpeedFactor(20)
```

グローバル速度比率を20%に設定します。

Speed

プロトタイプ：

```
Speed(R)
```

説明：

関節移動方式の速度比率を設定します。ロボットアームの実際の移動速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

必須パラメータ：

R：速度比率。値範囲：0~100。

例：

```
Speed(20)
Go(P1)
```

ロボットアームはソフトウェア設定速度の20%の速度比率で点P1に移動します。

Accel

プロトタイプ：

```
Accel(R)
```

説明：

関節移動方式の加速度比率を設定します。ロボットアームの実際の移動加速度比率は、パラメータ設定の比率に、制御ソフトウェア再現設定内の値を乗算し、さらにグローバルレートを乗算したものとします。

必須パラメータ：

R：加速度比率。値範囲：0~100。

例：

```
Accel(50)
Go(P1)
```

ロボットアームはソフトウェア設定速度の50%の加速度比率で点P1に移動します。

SpeedS

プロトタイプ：

```
SpeedS(R)
```

説明：

直線と円弧の移動方式の空間移動 (X/Y/Z) の速度比率を設定します。

必須パラメータ：

R：速度比率。値範囲：0~100。

例：

```
SpeedS(20)
Move(P1)
```

ロボットアームはソフトウェア設定速度の20%の速度比率で点P1に移動します。

AccelS

プロトタイプ：

```
AccelS(R)
```

説明：

直線と円弧の移動方式の空間移動 (X/Y/Z) の加速度比率を設定します。

必須パラメータ：

R：加速度比率。値範囲：0~100。

例：

```
AccelS(50)
Move(P1)
```

ロボットアームはソフトウェア設定速度の50%の加速度比率で点P1に移動します。

SpeedR

プロトタイプ：

```
SpeedR(R)
```

説明：

直線と円弧の移動方式の姿勢移動 (RX/RY/RZ) の速度比率を設定します。

必須パラメータ：

R：速度比率。値の範囲：1～100。

例：

```
SpeedR(20)
Move(P1)
```

ロボットアームはソフトウェア設定速度の20%の速度比率で点P1に移動します。

AccelR

プロトタイプ：

```
AccelR(R)
```

説明：

直線と円弧の移動方式の姿勢移動 (RX/RY/RZ) の加速度比率を設定します。

必須パラメータ：

R：加速度比率。値の範囲：1～100。

例：

```
AccelR(50)
Move(P1)
```

ロボットアームはソフトウェア設定速度の50%の加速度比率で点P1に移動します。

GetPose

プロトタイプ：

```
GetPose()
```

説明：

デカルト座標系におけるロボットアームのリアルタイムポーズを取得します。

オプションのパラメータ：

- user：ポーズ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：ポーズ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、グローバル工具座標系を使用します。

戻る：

ロボットアームの現在ポーズのデカルト座標値。

例：

```
local currentPose = GetPose()
Go(P1)
Go(currentPose)
```

ロボットアームはまずP1に移動してから、現在のポーズに戻ります。

GetAngle

プロトタイプ：

```
GetAngle()
```

説明：

関節座標系におけるロボットアームのリアルタイムポーズを取得します。

戻る：

ロボットアームの現在ポーズの関節座標値。

例：

```
local currentAngle = GetAngle()
Go(P1)
MoveJ(currentAngle)
```

ロボットアームはまずP1に移動してから、現在のポーズに戻ります。

CheckGo

プロトタイプ：

```
CheckGo(P)
```

説明：

関節移動インストラクションの実行可能性をチェックします。

必須パラメータ：

P：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。

戻る：

チェック結果。

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

例：

```
local status=CheckGo (P1)
if(status==0)
    then
        Go(P1)
    end
```

関節移動デフォルト設定でP1に到達できるかどうかをチェックし、到達できるならば、P1に関節移動します。

CheckMove

プロトタイプ：

```
CheckMove(P)
```

説明：

直線移動インストラクションの実行可能性をチェックします。

必須パラメータ：

P：目標点を表します。該プロジェクトの「ティーチング点」ページで追加した後にここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標の点位置のみに対応します。

戻る：

チェック結果。

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

例：

```
local status=CheckMove (P1)
if(status==0)
    then
        Move(P1)
    end
```

直線移動デフォルト設定でP1に到達できるかどうかをチェックし、到達できるならば、P1に直線移動します。

TCPSpeed

プロトタイプ

```
TCPSpeed(vt)
```

説明

絶対速度を設定します。この命令以降のデカルト座標系移動命令は設定された絶対速度で動作し、関節座標系移動命令には影響しません。TCPSpeedを設定すると、SpeedSは有効になりませんが、最大速度は引き続きグローバル速度（低減モードを含む）によって制限されます。

溶接プロセスパッケージを使用する場合、この命令が溶接関連の命令と矛盾する場合は、溶接命令が優先されます。

必須パラメータ

vt：絶対速度、単位：mm/s、値の範囲：(0,100000]

例

```
TCPSpeed(100)
Move(P1)
```

ロボット アームは、絶対速度100mm/sでP1まで直線移動します。

TCPSpeedEnd

プロトタイプ

```
TCPSpeedEnd()
```

説明

TCP Speed命令と組み合わせて使用し、絶対速度設定をオフにします。

例

```
TCPSpeed(100)
Move(P1)
TCPSpeedEnd()
Move(P2)
```

ロボットアームは絶対速度100mm/sでP1まで直線移動した後、グローバル速度でP2まで直線移動します。

InverseSolution

プロトタイプ

```
InverseSolution(P,User,Tool,isJointNear,JointNear)
```

説明

逆運動学の数値演算を実行する：ロボットアーム末端の与えられたデカルト座標系中の座標値が与えられ、ロボットアームの各関節の角度を計算します。

デカルト座標はTCPの空間座標と傾斜角のみを定義するので、ロボットアームは多種の異なる姿勢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。関節座標の設定の詳細については、isJointNearとJointNearパラメータを参照してください。

必須パラメータ

- P：目標のデカルト点を示します。
- User：校正されたユーザー座標系インデックス
- Tool：校正された工具座標系インデックス

オプションのパラメータ

- isJointNear：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。0または持っていない場合は、JointNearが無効であることを意味し、システムはロボットアームの現在の関節角度に応じて最も近い解を選択します。1場合は、JointNearに応じて最も近い解を選択します。
- JointNear：最も近い解の関節座標を選択するためのものです。

点の関節座標値を返します

。

例

```
InverseSolution({473.000000,-141.000000,469.000000,-180.000000,0.000,-90.000},0,0)
```

ロボットアームの先端のユーザー座標系0と関節座標系0でのデカルト座標は、{473,-141,469,-180,0,-90}です。関節座標を計算し、ロボットアームの現在の関節角度の最も近い解を選択します。

```
InverseSolution({473.000000,-141.000000,469.000000,-180.000000,0.000,-90.000},0,0,1,{0,0,-90,0,90,0})
```

ロボットアームの先端のユーザー座標系0および関節座標系0でのデカルト座標は、{473,-141,469,-180,0,-90}です。関節座標を計算し、{0,0,-90,0,90,0}の最も近い解を選択します。

SetBackDistance

プロトタイプ：

```
SetBackDistance(distance)
```

説明：

ロボットアームが衝突を検出してから元のルートに戻るまでの距離を設定します(衝突後の処理方法が復帰モードに設定されている場合のみ有効)。この命令で設定した値は、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：

```
SetBackDistance(20)
```

衝突して戻る距離を20mmで設定します。

相対移動インストラクション

移動命令関数はロボットアームをオフセット移動を行うように制御するために使用されます。このグループの命令はすべてキュー命令です。

RP

プロトタイプ：

```
RP(P, {OffsetX, OffsetY, OffsetZ})
```

説明：

指定した点位置に対してデカルト座標系におけるX、Y、Z方向上のオフセット量を追加し、新たなデカルト座標点を戻します。

必須パラメータ：

- オフセット前の点位置。該プロジェクトの「ティーチング点」ページで追加してからここで引用してもよく、点位置をカスタマイズしてもよいが、デカルト座標点位置のみに対応します。
- OffsetX、OffsetY、OffsetZ：デカルト座標系におけるX、Y、Z方向上のオフセット量。単位：mm。

戻る：

オフセット後のデカルト座標点。

例：

```
Go(RP(P1, {30,50,10}))
```

P1をX、Y、Z軸上でそれぞれ一定の距離オフセットさせてから、オフセット後の点位置に移動させます。

RJ

プロトタイプ：

```
RJ(P, {Offset1, Offset2, Offset3, Offset4, Offset5, Offset6})
```

説明：

指定した点位置に対して関節座標系においてJ1~J6軸オフセット量を追加し、新たな関節座標点を戻します。

必須パラメータ：

- オフセット前の点位置。該プロジェクトの「ティーチング点」ページで追加してからここで引用してもよく、点位置をカスタマイズしてもよいが、関節座標点位置のみに対応します。
- Offset1~Offset6：関節座標系におけるJ1軸 ~ J6軸方向上のオフセット量。単位：度。

戻る：

オフセット後の関節座標点。

例：

```
MoveJ(RJ(P1, {60,50,32,30,25,30}))
```

P1をJ1~J6軸上でそれぞれ一定の角度オフセットさせてから、オフセット後の点位置に移動させます。

GoToolR

プロトタイプ：

```
GoToolR({x, y, z, rx, ry, rz}, Tool, "User=0 CP=1 Speed=50 Accel=20 SYNC=1")
```

説明：

現在の位置から、指定工具座標系に沿って、関節移動方式で相対移動を行います。関節移動の軌跡は直線ではなく、すべての関節は同時に移動を完了します。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定工具座標系におけるオフセット量。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°

- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。

SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
GoToolR({10, 10, 10, 0, 0, 0},0)
```

ロボットアームはデフォルト設定で、工具座標系0に沿って指定のオフセット点まで関節移動します。

MoveToolR

プロトタイプ：

```
MoveToolR({x, y, z, rx, ry, rz}, Tool, "User=0 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

説明：

現在の位置から、指定工具座標系に沿って、直線移動方式で相対移動を行います。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定工具座標系におけるオフセット量。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°

- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
MoveToolR({10, 10, 10, 0, 0, 0},0)
```

ロボットアームはデフォルト設定で、工具座標系0に沿って指定のオフセット点まで直線移動します

GoUserR

プロトタイプ：

```
GoUserR({x, y, z, rx, ry, rz}, User, "Tool=0 CP=1 Speed=50 Accel=20 SYNC=1")
```

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}：目標点の現在の位置に対して指定ユーザー座標系におけるオフセット量。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°
- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。

オプションパラメータ：

- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。 SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
GoUserR({10, 10, 10, 0, 0, 0},0)
```

ロボットアームはデフォルト設定で、ユーザー座標系0に沿って指定のオフセット点まで関節移動します。

MoveUserR

プロトタイプ：

```
MoveUserR({x, y, z, rx, ry, rz}, User, "Tool=0 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

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}：目標点の現在の位置に対して指定ユーザー座標系におけるオフセット量。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°
- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。

オプションパラメータ：

- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。 SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
MoveUserR({10, 10, 10, 0, 0, 0},0)
```

ロボットアームはデフォルト設定で、ユーザー座標系0に沿って指定のオフセット点まで直線移動します。

MoveJR

プロトタイプ：

```
MoveJR({Offset1, Offset2, Offset3, Offset4, Offset5, Offset6},"User=1 Tool=2 CP=1 Speed=50 Accel=20 SYNC=1")
```

説明：

現在位置から、ポイントツーポイント（関節移動）方式で関節オフセット角度に移動します。

必須パラメータ：

Offset1~Offset6：関節座標系におけるJ1軸 ~ J6軸方向上のオフセット量。単位：度。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。

- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
MoveJR({20,20,10,0,10,0})
```

ロボットアームはデフォルト設定に従ってオフセット角度に関節移動します。

GoR

プロトタイプ：

```
GoR({OffsetX, OffsetY, OffsetZ},"User=1 Tool=2 CP=1 Speed=50 Accel=20 SYNC=1")
```

説明：

该指令已被**GoToolR**和**GoUserR**取代。

現在位置から、ポイントツーポイント（関節移動）方式でデカルト座標系におけるオフセット位置に移動します。関節移動の軌跡は非線形で、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：

OffsetX、OffsetY、OffsetZ：デカルト座標系におけるX軸、Y軸、Z軸方向上のオフセット。単位：mm。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- Speed：移動速度比率。値範囲：1~100。
- Accel：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
GoR({10,10,10})
```

ロボットアームはデフォルト設定に従ってポイントツーポイント方式でオフセット点に移動します。

MoveR

プロトタイプ：

```
MoveR({OffsetX, OffsetY, OffsetZ},"User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

説明：

この命令は**MoveToolR**と**MoveUserR**に置き換えられました。

現在位置から、デカルト座標系におけるオフセット位置に直線移動します。

必須パラメータ：

OffsetX、OffsetY、OffsetZ：デカルト座標系におけるX軸、Y軸、Z軸方向上のオフセット。単位：mm。

オプションパラメータ：

- User：ユーザー座標系インデクス。設定で追加した後にここで引用する必要があります。
- Tool：ツール座標系インデクス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCPインストラクションを参照）。値範囲：0~100。
- SpeedS：移動速度比率。値範囲：1~100。
- AccelS：移動加速度比率。値範囲：1~100。
- SYNC：同期フラグ、値範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出した後に、インストラクションの実行が完了しないと戻りません。

例：

```
MoveR({10,10,10})
```

ロボットアームはデフォルト設定に従ってオフセット点に直線移動します。

I/O

I/Oインストラクションは、ロボットアームシステムI/Oの読み書きと関連パラメータを設定するためのものです。

DI

プロトタイプ：

```
DI(index)
```

説明：

即時命令。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。

戻る：

対応するDI端子のレベル（ON/OFF）。

例：

```
if (DI(1)==ON) then
Move(P1)
end
```

DI1がハイレベル（ON）である場合にロボットアームは直線移動の方式でP1点まで移動します。

DIGroup

プロトタイプ：

```
DIGroup(index1,...,indexN)
```

説明：

即時命令。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。

戻る：

対応するDI端子の状態（ON/OFF）を配列で返します。

例：

```
local digroup = DIGroup(1,2)
if (digroup[1]&digroup[2]==ON)
then
    Move(P1)
end
```

DI1 と DI2 が両方とも ON の場合、ロボットアームは直線移動の方式でP1点まで移動します。

WaitDI

プロトタイプ：

```
WaitDI(index,ON|OFF,period)
```

説明：

即時命令。デジタル入力ポート状態を読み取り、読み取った状態が指定した状態と一致する場合、プログラムは次を実行します。そうでない場合、指定されたサイクルの間隔で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。
- ON|OFF：待機するDIポートの状態で、ONはハイレベル、OFFはローレベルです。

オプションパラメータ：

period：読み取りサイクルで、単位はms、デフォルトは50msです。

例：

```
WaitDI(1,ON,50)
Move(P1)
```

50msごとにDI1の状態を読み取ります。DI1がハイレベルである場合、プログラムは次を実行します。ロボットアームは直線運動の方式でP1点まで移動します。

DO

プロトタイプ：

```
DO(index,ON|OFF)
```

説明：

キュー命令。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ポートの状態で、ONはハイレベル、OFFはローレベルです。

例：

```
DO(1,ON)
```

DO1をハイレベルに設定します（ON）。

DOExecute

プロトタイプ：

```
DOExecute(index,ON|OFF)
```

説明：

即時命令。現在のインストラクションキューに関係なく、直ちに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ポートの状態で、ONはハイレベル、OFFはローレベルです。

例：

```
DOExecute(1,ON)
```

現在のインストラクションキューに関係なく、直ちにDO1をハイレベル（ON）に設定します。

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。

例：

```
DOGroup({1,ON},{2,ON})
```

DO1とDO2をONに設定します。

ToolDO

プロトタイプ：

```
ToolDO(index,ON|OFF)
```

説明：

キュー命令。先端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：先端のDO端子の番号。
- ON|OFF：設定するDOポートの状態で、ONはハイレベル、OFFはローレベルです。

例：

```
ToolDO(1,ON)
```

先端のDO1をハイレベル（ON）に設定します。

ToolDOExecute

プロトタイプ：

```
ToolDOExecute(index,ON|OFF)
```

説明：

即時命令。現在のインストラクションキューに関係なく、直ちに先端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：先端のDO端子の番号。
- ON|OFF：設定するDOポートの状態で、ONはハイレベル、OFFはローレベルです。

例：

```
ToolDOExecute(1,ON)
```

現在のインストラクションキューに関係なく、直ちに先端のDO1をハイレベル（ON）に設定します。

ToolDI

プロトタイプ：

```
ToolDI(index)
```

説明：

即時命令。先端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：先端のDI端子の番号。

戻る：

対応するDI端子のレベル（ON/OFF）。

例：

```
if (ToolDI(1)==ON) then
Move(P1)
end
```

先端DI1がハイレベル（ON）である場合にロボットアームは直線移動の方式でP1点まで移動します。

ToolAI

プロトタイプ：

```
ToolAI(index)
```

説明：

即時命令。先端のアナログ入力ポートの値を読み取ります。使用する前に、ToolAnalogModeにより端子を電圧入力モードに設定してください。

必須パラメータ：

index：先端のAI端子の番号。

戻る：

対応するAI端子の値。

例：

```
test = ToolAI(1)
```

先端AI1の値を読み取り、変数testに割り当てられます。

ToolAnalogMode

プロトタイプ：

```
ToolAnalogMode(mode)
```

説明：

即時命令。先端のアナログ入力ポートのモードを設定します。

必須パラメータ：

mode：アナログ入力ポートのモード

- 00：デフォルト、485モード。
- 10：電流収集モード。
- 11：0~3.3V 電圧入力モード。
- 12：0~10V 電圧入力モード。

例：

```
ToolAnalogMode(11)
```

先端のアナログ入力ポートのモードを0~3.3V電圧入力モードに設定します。

AO

プロトタイプ：

```
AO(index,value)
```

説明：

キュー命令。アナログ出力ポートの電圧値を設定します。

必須パラメータ：

- index：AO端子の番号。
- value：設定する電圧値で、値範囲は0~10です

例：

```
AO(1,2)
```

AO1の電圧を2Vに設定します。

AOExecute

プロトタイプ：

```
AOExecute(index,value)
```

説明：

即時命令。現在のインストラクションキューに関係なく、直ちにアナログ出力ポートの電圧値を設定します。

必須パラメータ：

- index：AO端子の番号。
- value：設定する電圧値で、値範囲は0~10です

例：

```
AOExecute(1,2)
```

現在のインストラクションキューに関係なく、直ちにAO1の電圧を2Vに設定します。

AI

プロトタイプ：

```
AI(index)
```

説明：

即時命令。アナログ入力ポートの値を読み取ります。

必須パラメータ：

index：AI端子の番号。

戻る：

対応するAI端子の値。

例：

```
test = AI(1)
```

AI1の値を読み取り、変数testに割り当てられます。

SetToolPower

プロトタイプ：

```
SetToolPower(status)
```

説明：

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

必須パラメータ：

status：エンドツールの電源状態。1：電源オフ、0：電源オン

例：

```
SetToolPower(1)
```

エンドエフェクタの電源を切断します。

SetABZPPC

プロトタイプ：

```
 SetABZPPC(value)
```

説明：

即時命令。エンコーダーの現在値を設定します。

必須パラメータ：

value：エンコーダーの位置、単位：パルス。

例：

```
SetABZPPC(1000)
```

ABZエンコーダーの値を1000パルスに設定します。

GetABZ

プロトタイプ：

```
GetABZ()
```

説明：

即時命令。設定したABZエンコーダーの分解能を取得します。

戻り値：

エンコーダーの分解能で、単位：パルス/ms。

例：

```
local abz = GetABZ()
```

設定したABZエンコーダーの分解能を取得し、変数abzに割り当てられます。

TCP/UDP

TCP/UDP関数は、TCPまたはUDP通信を行うために使用されます。このグループの命令はすべて即時命令です。

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を作成します。1つのみを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アームのIPアドレスであり、クライアント作成時は相手側のアドレスです。
- port：サーバーポート。ロボットアームがサーバーとして機能する場合、システムがすでに占有している次のポートを使用しないでください：

  22、23、502（0～1024の間のポートはLinuxシステムのカスタムポートであり、占有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）、

  30005, 30006, 5000~5004, 6000, 8080, 11000, 11740, 22000, 22002, 29999, 30003, 30004, 60000, 65500~65515

戻る：

- err：0はTCPネットワークオブジェクトの作成に成功したことを表し、1はTCPネットワークオブジェクトの作成に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**：

```
local ip="192.168.5.1" -- ロボットアームのIPアドレスは、サーバーのIPアドレスとする
local port=6001 -- サーバーポート
local err=0
local socket=0
err, socket = TCPCreate(true, ip, port)
```

TCPサーバーを作成します。

例**2**：

```
local ip="192.168.5.25" -- カメラのような外部設備のIPアドレスは、サーバーのIPアドレスとする
local port=6001 -- サーバーポート
local err=0
local socket=0
err, socket = TCPCreate(false, ip, port)
```

TCPクライアントを作成します。

TCPStart

プロトタイプ：

```
TCPStart(socket, timeout)
```

説明：

TCP接続を確立します。ロボットアームをサーバーとするとき、クライアントが接続するのを待機します。ロボットアームをクライアントとするとき、サーバーに自発的に接続します

必須パラメータ：

- socket：作成したsocketオブジェクト。
- timeout：待機タイムアウト時間。単位：秒。0に設定した場合、接続の確立が成功するまで待機します。0ではない場合、設定した時間を超えると接続に失敗したと戻します。

戻る：

接続結果。

- 0：接続に成功しました
- 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：

```
err = TCPStart(socket, 0)
```

TCP接続の確立を開始し、接続確立が成功するまで待機します。

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。

オプションパラメータ：

- timeout：待機タイムアウト時間。単位：秒。0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定していない場合、RecBufバッファ形式はtable形式とします。「string」に設定した場合、RecBufバッファは文字列です。

戻る：

- err：0はデータの受信に成功したことを表し、1はデータの受信に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：

```
err, RecBuf = TCPRead(socket,0,"string") -- RecBufデータタイプは文字列です
err, RecBuf = TCPRead(socket, 0) -- RecBufデータタイプはtableです
```

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。

TCPWrite

プロトタイプ：

```
TCPWrite(socket, buf, timeout)
```

説明：

データをTCP相手側に送信します。

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。

オプションパラメータ：

timeout：待機タイムアウト時間。単位：秒。0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

送信結果。

- 0：送信に成功しました。
- 1：送信失敗

例：

```
TCPWrite(socket, "test")
```

TCPデータを送信します。データコンテンツは「test」とします。

TCPDestroy

プロトタイプ：

```
TCPDestroy(socket)
```

説明：

TCP接続を切断し、socketオブジェクトを廃棄します。

必須パラメータ：

socket：作成したsocketオブジェクト。

戻る：

実行結果。

- 0：実行に成功しました
- 1：実行に失敗しました

例：

```
TCPDestroy(socket)
```

TCP相手側との接続を切断します。

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を作成します。1つのみを作成することができます。

必須パラメータ：

- isServer：固定的にfalseを記入します。
- IP：相手側IPアドレス。ロボットアームIPアドレスと同じネットワークセグメントにあり、かつ競合しない必要があります。
- port：相手側ポート。

戻る：

- err：0はUDPネットワークオブジェクトの作成に成功したことを表し、1はUDPネットワークオブジェクトの作成に失敗したことを表します。
- socket：作成したsocketオブジェクト。

例：

```
local ip="192.168.5.25" -- カメラのような外部設備のIPアドレスは、ピアのIPアドレスとする
local port=6001 -- ピアポート
local err=0
local socket=0
err, socket = UDPCreate(false, ip, port)
```

UDPネットワークオブジェクトを作成します。

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。

オプションパラメータ：

- timeout：待機タイムアウト時間。単位：秒。0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定していない場合、RecBufバッファ形式はtable形式とします。「string」に設定した場合、RecBufバッファは文字列です。

戻る：

- err：0はデータの受信に成功したことを表し、1はデータの受信に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：

```
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
```

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。

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。

オプションパラメータ：

timeout：待機タイムアウト時間。単位：秒。0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

送信結果。

- 0：送信に成功しました。
- 1：送信失敗

例：

```
UDPWrite(socket, "test")
```

UDPデータを送信します。データコンテンツは「test」とします。

Modbus

Modbus関数はModbus通信を行うために使用されます。このグループの命令はすべて即時命令です。

ModbusCreate

プロトタイプ：

```
ModbusCreate(IP,port,slave_id)
```

説明：

Modbusマスターを作成し、スレーブと接続を確立します。

オプションパラメータ：

- IP：スレーブのIPアドレス。指定しない、あるいは「127.0.0.1」または「0.0.0.1」に指定した場合、本機に接続するModbusスレーブを示します。
- port：スレーブのポート。
- slave_id：スレーブID。
- 値の範囲は0～4です。

- isRTU：空または0の場合、ModbusTCP通信が確立されていることを示します。1の場合はModbusRTU通信が確立していることを意味します。

> ⚠ 注意：
> このパラメータは、接続確立後のデータ転送に使用するプロトコル形式を決定しますが、接続結果には影響しません。したがって、マスターを作成する際にこのパラメータを誤って設定した場合でも、作成は成功しますが、その後の通信では異常が発生する可能性があります。

戻る：

- err：
  - 0：Modbusマスターの作成に成功しました
  - 1：作成したマスターが既に5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：マスターのインデックスで、値範囲は0~4です。

例**1**：

```
local ip="192.168.5.123" -- スレーブのIPアドレス
local port=503 -- スレーブのポート
local err=0
local id=0
err, id = ModbusCreate(ip, port, 1)
```

Modbusマスターを作成し、指定されたスレーブと接続を確立します。

例**2**：

すべてのパラメータを指定しない場合、あるいはIPを「127.0.0.1」または「0.0.0.1」に指定した場合は、本機に接続するModbusスレーブを示します。以下の命令の使用方法を参照してください。

```
ModbusCreate()
```

```
ModbusCreate("127.0.0.1")
```

```
ModbusCreate("0.0.0.1")
```

```
ModbusCreate("127.0.0.1", xxx,xxx) -- xxxは任意の値
```

```
ModbusCreate("0.0.0.1", xxx,xxx) -- xxxは任意の値
```

例**3**：

```
err,id=ModbusCreate("127.0.0.1",60000,1,1)
```

上記の例は特別な使用法であり、ロボットアームの端にある485ポートを介した外部Modbus RTUスレーブとの通信を示しています。

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。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：接点レジスタの開始アドレスのアドレス。値範囲：0~9999。
- count：連続で読み取った値の数。端末経由でスレーブと通信する場合の最大値は216（CR）または984（Nova）、それ以外の場合の最大値は2008です。

戻る：

接点レジスタのアドレスの値は、tableに保存されます。tableの1番目の値は接点レジスタの開始アドレスの値に対応します。

例：

```
inBits = GetInBits(id,0,5)
```

アドレス0から始まり、連続で5個のアドレスの値を読み取ります。

GetInRegs

プロトタイプ：

```
GetInRegs(id, addr, count, type)
```

説明：

指定されたデータタイプに基づき、Modbusスレーブの入力レジスタアドレスの値を読み取ります。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：入力レジスタの開始アドレスのアドレス。値範囲：0~9998。
- count：連続で読み取った値の数。

オプションパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの符号がない整数（2バイト、1レジスタ占用）。端末経由でスレーブと通信する場合は最大13個（CR）または61個（Nova）、その他の場合は最大125個の値を連続して読み取ることができます。

- U32：32ビットの符号がない整数（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）。端末経由でスレーブと通信する場合は最大3個（CR）または15個（Nova）、その他の場合は最大31個の値を連続して読み取ることができます。

戻る：

入力レジスタのアドレスの値は、tableに保存されます。tableの1番目の値は、入力レジスタの開始アドレスの値に対応します。

例：

```
data = GetInRegs(id, 2048, 1, “U32”)
```

アドレス2048から始まり、1つの32ビットの符号がない整数を読み取ります。

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。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：コイルレジスタの開始アドレスのアドレス。値範囲：0~9999。
- count：連続で読み取った値の数。端末経由でスレーブと通信する場合の最大値は216（CR）または984（Nova）、それ以外の場合の最大値は2008です。

戻る：

コイルレジスタのアドレスの値は、tableに保存されます。table中の1番目の値は、コイルレジスタの開始アドレスの値に対応します。

例：

```
Coils = GetCoils(id,0,5)
```

アドレス0から始まり、連続で5個のアドレスの値を読み取ります。

GetHoldRegs

プロトタイプ：

```
GetHoldRegs(id, addr, count, type)
```

説明：

指定されたデータタイプに基づき、Modbusスレーブの保持レジスタアドレスの値を読み取ります。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：保持レジスタの開始アドレスのアドレス。値範囲：0~9998。
- count：連続で読み取った値の数。

オプションパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの符号がない整数（2バイト、1レジスタ占用）。端末経由でスレーブと通信する場合は最大13個（CR）または61個（Nova）、その他の場合は最大125個の値を連続して読み取ることができます。
- U32：32ビットの符号がない整数（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大6個（CR）または30個（Nova）、その他の場合は最大62個の値を連続して読み取ることができます。
- F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）。端末経由でスレーブと通信する場合は最大3個（CR）または15個（Nova）、その他の場合は最大31個の値を連続して読み取ることができます。

戻る：

保持レジスタのアドレスの値は、tableに保存されます。tableの1番目の値は、保持レジスタの開始アドレスの値に対応します。

例：

```
data = GetHoldRegs(id, 2048, 1, “U32”)
```

アドレス2048から始まり、1つの32ビットの符号がない整数を読み取ります。

SetCoils

プロトタイプ：

```
SetCoils(id, addr, count, table)
```

説明：

指定した値をコイルレジスタが指定するアドレスに書き込みます。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：コイルレジスタの開始アドレスのアドレス。値範囲：6~9999。
- count：連続で書き込んだ値の数。端末経由でスレーブと通信する場合の最大値は440、それ以外の場合の最大値は1976です。
- table：コイルレジスタに書き込む値は、tableに保存されます。table中の1番目の値は、コイルレジスタの開始アドレスの値に対応します。

例：

```
local Coils = {0,1,1,1,0}
SetCoils(id, 1024, #coils, Coils)
```

アドレス1024から始まり、連続で5個のコイルを書き込みます。

SetHoldRegs

プロトタイプ：

```
SetHoldRegs(id, addr, count, table, type)
```

説明：

指定されたデータタイプに基づき、指定した値を保持レジスタが指定するアドレスに書き込みます。

必須パラメータ：

- id：Modbusマスターを作成する場合に返すマスターインデックス。
- addr：保持レジスタの開始アドレスのアドレス。値範囲：0~9998。

- count：連続で書き込んだ値の数。
- table：コイルレジスタに書き込む値は、tableに保存されます。table中の1番目の値は、コイルレジスタの開始アドレスの値に対応します。

オプションパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの符号がない整数（2バイト、1レジスタ占用）。端末経由でスレーブと通信する場合は最大27個、その他の場合は最大123個の値を連続して書き込むことができます。
- U32：32ビットの符号がない整数（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大13個、その他の場合は最大61個の値を連続して書き込むことができます。
- F32：32ビットの単精度浮動小数点（4バイト、2レジスタ占用）。端末経由でスレーブと通信する場合は最大13個、その他の場合は最大61個の値を連続して書き込むことができます。
- F64：64ビットの倍精度浮動小数点（8バイト、4レジスタ占用）。端末経由でスレーブと通信する場合は最大6個、その他の場合は最大30個の値を連続して書き込むことができます。

例：

```
local data = {95.32105}
SetHoldRegs(id, 2048, #data, data, “F64”)
```

アドレス2048から始まり、1つの64ビットの倍精度浮動小数点を書き込みます。

ModbusClose

プロトタイプ：

```
ModbusClose(id)
```

説明：

Modbusスレーブとの接続を解除します。

オプションパラメータ：

id：Modbusマスターを作成する場合に返すマスターインデックス。

戻る：

操作結果

- 0：接続解除成功。
- 1：接続解除失敗。

例：

```
ModbusClose(id)
```

Modbusスレーブとの接続を解除します。

プログラム制御

プログラム制御関数はプログラム実行制御に関連する汎用関数です。ここで、while、if、forはluaのフロー制御インストラクションです。Lua基本文法 - フロー制御を参照してください。printはプリントインストラクションです。コンソールに制御情報を出力するために使用されます。

print

プロトタイプ：

```
print(var)
```

説明：

即時命令。指定された情報をコンソールにプリントアウトします。

必須パラメータ：

var：情報をプリントアウトします

例：

```
local var = "Hello World"
print(var)
```

Hello Worldをコンソールにプリントアウトします。

Sleep

プロトタイプ：

```
Sleep(time)
```

説明：

即時命令。スクリプトは次の命令の実行を遅らせます。この命令は、スクリプト全体の実行をブロックするために使用されます。

必須パラメータ：

time：遅延時間。単位：ミリ秒。パラメータには小数点を使用できません。アラームや命令が無効になります。0.5 * 1000 という形式も使用できません。パラメータに整数を使用してください。

例：

```
DO(1,ON)
Sleep(100)
DO(1,OFF)
```

DO1をONに設定し、100ms待機してからDO1をOFFに設定します。

Wait

プロトタイプ：

```
Wait(time)
```

説明：

キュー命令。移動命令の発行を遅延させ、または移動が完了した後に次の命令の発行を遅延させます。この命令は、バックグラウンドアルゴリズムキューの実行をブロックするだけです。

必須パラメータ：

time：遅延時間。単位：ミリ秒。パラメータには小数点を使用できません。アラームや命令が無効になります。0.5 * 1000 という形式も使用できません。パラメータに整数を使用してください。

例：

```
DO(1,ON)
Wait(100)
Go(P1)
Wait(100)
DO(1,OFF)
```

DO1をONに設定した後に、100ms待機してからロボットアームがP1に移動し、P1に実行した後に100ms待機してから、DO1をOFFに設定します。

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操作が必要です。

例：

```
Go(P1)
Pause()
Go(P2)
```

ロボットアームは点P1に移動した後に実行を一時停止し、外部制御によって実行を続行させなければ点P2までに移動しません。

SetCollisionLevel

プロトタイプ：

```
SetCollisionLevel(level)
```

説明：

キュー命令。衝突検出レベルを設定します。該インターフェスを介して設定した衝突検出レベルは、プロジェクト実行期間中にのみ有効で、プロジェクト停止した後に修正前の値に戻ります。

選択必須パラメータ：

level：衝突検出レベル。値範囲は0~5とします。0は衝突検出をオフにすることを表します。1~5はレベルが高いほど衝突検出の感度が高い。

例：

```
SetCollisionLevel(2)
```

衝突検出レベルをレベル2に設定します。

ResetElapsedTime

プロトタイプ：

```
ResetElapsedTime()
```

説明：

キュー命令。このインストラクションの前のすべてのインストラクションの実行が完了するのを待機してから時間計測を開始します。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されます。

例：

ElapsedTimeの例を参照してください。

ElapsedTime

プロトタイプ：

```
ElapsedTime()
```

説明：

キュー命令。時間計測を終了するとき、時間差を戻します。ElapsedTime()と合わせて使用する必要があります。

このポートは35分以内の時間差のみをカウントできるため、時間差が35分を超えると、このポートは誤った時間差を返します。時間差が長い場合は、Systimeポートを使用して計算することをお勧めします。

戻す：

時間計測の開始から時間計測の終了までの時間差。単位：ミリ秒。

例：

```
Go(P2, "Speed=100 Accel=100")
ResetElapsedTime()
for i=1,10 do
Move(P1)
Move(P2)
end
print (ElapsedTime())
```

ロボットアームがP1とP2の間で10回往復移動する時間を計算し、コンソールにプリントします。

Systime

プロトタイプ：

```
Systime()
```

説明：

即時命令。現在のシステム時間を取得します。

戻す：

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

例：

```
local time1 = Systime()
Move(P1)
local time2 = Systime()
print(time2-time1)
```

ロボットアームがP1に移動するのにかかる時間（ミリ秒）を計算します。

SetUser

プロトタイプ：

```
SetUser(index,table)
```

説明：

キュー命令。指定したユーザー座標系を修正します。この変更は現在のプロジェクト実行中にのみ有効で、プロジェクトが停止後、座標系は変更前の値に戻ります。

必須パラメータ：

- index：ユーザー座標系番号。値範囲は0~9とします。ここで、座標系0はデフォルトのユーザー座標系とします。
- table：修正後のユーザー座標系行列。形式は{x, y, z, rx, ry, rz}とします。計算にはCalcUserを使用することをお勧めします。

例：

```
SetUser(1,{10,10,10,0,0,0})
```

ユーザー座標系1をX=10、Y=10、Z=10、RX=0、RY=0、RZ=0に修正します。

CalcUser

プロトタイプ：

```
CalcUser(index,matrix_direction,table)
```

説明：

キュー命令。ユーザー座標系を計算します。

必須パラメータ：

- index：ユーザー座標系のインデックス。値の範囲：[0,9]。ユーザー座標系0の初期値は基座標系です。
- 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：ユーザー座標系のオフセット値で、形式は{x, y, z, rx, ry, rz}。

戻り値：

計算で得られたユーザー座標系で、形式は{x, y, z, rx, ry, rz}。

例：

```
-- 以下の計算過程は次のように等価です：初期ポーズがユーザー座標系1と同じ座標系が、基座標系に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newUserが得られます。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
```

```
-- 以下の計算過程は次のように等価です：初期ポーズがユーザー座標系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}だけ回転すると、新しい座標系newUserが得られます。
newUser = CalcUser(1,0,{10,10,10,10,10,10})
```

SetTool

プロトタイプ：

```
SetTool(index,table)
```

説明：

キュー命令。指定したツール座標系を修正します。この変更は現在のプロジェクト実行中にのみ有効で、プロジェクトが停止後、座標系は変更前の値に戻ります。

必須パラメータ：

- index：ツール座標系番号。値範囲は0~9とします。ここで、座標系0はデフォルトのツール座標系とします。
- table：修正後のツール座標系行列。形式は{x, y, z, rx, ry, rz}とします。計算にはCalcToolを使用することをお勧めします。

例：

```
SetTool(1,{10,10,10,0,0,0})
```

ツール座標系1をX=10、Y=10、Z=10、RX=0、RY=0、RZ=0に修正します。

CalcTool

プロトタイプ：

```
CalcTool(index,matrix_direction,table)
```

説明：

キュー命令。工具座標系を計算します。

必須パラメータ：

- index：工具座標系のインデックス。値の範囲：[0,9]。座標系0の初期値はフランジ座標系（TCP0）に基づいています。
- matrix_direction：計算の方向。
  - 1：左乗算。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：工具座標系で、形式は{x, y, z, rx, ry, rz}。

戻り値：

計算で得られた工具座標系で、形式は{x, y, z, rx, ry, rz}。

例：

```
-- 以下の計算過程は次のように等価です：初期ポーズが工具座標系1と同じ座標系が、フランジ座標系（TCP0）に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newToolが得られます。
newTool = CalcTool(1,1,{10,10,10,10,10,10})
```

```
-- 以下の計算過程は次のように等価です：初期ポーズが工具座標系1と同じ座標系が、工具座標系1に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newToolが得られます。
newTool = CalcTool(1,0,{10,10,10,10,10,10})
```

LoadSet

プロトタイプ

```
LoadSet(weight,inertia)
```

説明

キュー命令。ロボットアームの先端負荷を設定します。この変更は現在のプロジェクトの実行中にのみ有効になり、プロジェクトが停止すると負荷パラメータは変更前の値に戻ります。

必須パラメータ：

- weight：負荷重量を設定し、値の範囲は各型番のロボットの負荷範囲を超えてはなりません。単位：kg
- inertia：負荷慣性モーメント、単位：kgm²

例：

```
LoadSet(3,0.4)
```

先端負荷重量を3kg、負荷慣性を0.4kgm²に設定します。

LoadSwitch

プロトタイプ

```
LoadSet(status)
```

説明

キュー命令。負荷設定を切り替えます。負荷設定はデフォルトではオフになっていますが、オンにすると衝突検知の感度が向上します

必須パラメータ：

status：負荷設定のスイッチ。0はオフ、1はオンを意味します。

例：

```
LoadSwitch(1)
```

負荷設定をオンにします。

軌跡

GetTraceStartPose

プロトタイプ：

```
GetTraceStartPose(track)
```

説明：

軌跡フィッティング（StartTrace）用の軌跡の最初の点を取得します。

必須パラメータ：

track：サフィックスを含む軌跡ファイル名。

戻り値：

軌跡の最初の点の座標。

StartTrace

プロトタイプ：

```
StartTrace(track, "User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")
```

説明：

軌跡フィッティング。移動軌跡は軌跡ファイルに記録された点をもとにフィッティングされて再現されますが、移動速度の計算方法は直線運動と同様であり、軌跡記録時の移動速度の影響を受けません。

必須パラメータ：

track：サフィックスを含む軌跡ファイル名。

オプションのパラメータ：

- User：ユーザー座標系インデックス。設定で追加した後にここで引用する必要があります。
- Tool：工具座標系インデックス。設定で追加した後にここで引用する必要があります。
- CP：移動時に設定した連続経路制御（詳細については移動パラメータ内のCP命令を参照）。値の範囲：0～100。
- SpeedS：移動速度比率。値の範囲：1～100。
- AccelS：移動加速度比率。値の範囲：1～100。

- SYNC：同期フラグ、値の範囲：0または1。デフォルト値は0とします。
  SYNC = 0は非同期実行を表し、呼び出した後にすぐ戻るが、実行状況は無視します。
  SYNC = 1は同期実行を表し、呼び出し後は命令が完全に実行されるまで戻りません。

例：

```
local track = "track.json"
local P1 = GetTraceStartPose(track)
Go(P1)
StartTrace(track)
```

track.jsonファイルに記録された軌跡をフィッティングして再現します。

GetPathStartPose

プロトタイプ：

```
GetPathStartPose(track)
```

説明：

軌跡再現（StartPath）用の軌跡の最初の点を取得します。

必須パラメータ：

track：サフィックスを含む軌跡ファイル名。

戻り値：

軌跡の最初の点の座標。

StartPath

プロトタイプ：

```
StartPath(track, "User=1 Tool=2 sample = 50 freq = 0.2 Multi=1 isConst=0 isCart=0 isRev=0 isRe
l=0")
```

説明：

軌跡フィッティング。移動軌跡は軌跡ファイルに記録された点をもとにフィッティングされて再現されますが、移動速度の計算方法は直線運動と同様であり、軌跡記録時の移動速度の影響を受けません。

必須パラメータ：

track：サフィックスを含む軌跡ファイル名。

オプションのパラメータ：

- User：ユーザー座標系インデックス。設定で追加した後にここで引用する必要があります。
- Tool：工具座標系インデックス。設定で追加した後にここで引用する必要があります。
- sample：軌跡点のサンプリング間隔です。つまり、軌跡ファイルを作成した場合の、隣接する2つの点のサンプリング時間差です。値の範囲：[8, 1000]、単位：ms、引数がない場合のデフォルト値は50です（コントローラが軌跡ファイルを記録する場合のサンプリング間隔）。
- freq：フィルタリング係数で、このパラメータの値が小さくなるほど、再現する軌跡曲線はスムーズになります。ただし、元の軌跡に対する変形が厳しくなるほど、元の軌跡の連続経路制御程度に基づいて適切なフィルタリング係数を設定してください。値の範囲：（0,1]、1はフィルタリング終了を示し、引数がない場合のデフォルト値は0.2です。
- Multi：再現する場合の速度倍数。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。
  - 0は元の速度で再現することを意味します。
  - 1は均一な速度で再現することを意味し、軌跡内の一時停止とデッドゾーンは削除されます。
- isCart：デカルト点を使用するかどうか。引数がない場合のデフォルト値は0です。
  - 0は、軌跡ファイル内の関節点を使用して軌跡を計画することを意味します。
  - 1は、軌道ファイル内のデカルト点を使用して軌跡を計画することを意味します。
- isRev：軌跡反転。引数がない場合のデフォルト値は0です。
  - 0は反転がないことを意味します。
  - 1は、軌跡ファイル内の終点を始点、始点を終点として使用して、軌跡が反転されることを意味します。
- isRel：軌跡オフセット。引数がない場合のデフォルト値は0です。
  - 0はオフセットがないことを意味します。
  - 1は軌跡のオフセットを意味し、ロボットアームの現在点を軌跡の始点とし、軌跡全体をオフセットします。

例：

```
local track = "track.json"
local P1 = GetPathStartPose(track)
Go(P1)
StartPath(track)
```

track.jsonファイルに記録された軌跡を元の速度で再現します。

ビジュアル

ビジュアルプロセスパックはカメラ通信関連の構成を構成するために使用されます。カメラはロボットの動作範囲内に固定し、位置が固定し、撮影視野範囲が固定し、カメラはロボットの眼を務めます。イーサネット通信またはI/Oトリガー方式によりロボットとデータインタラクションを行います。

カメラの取付と構成方法はカメラによって異なりますので、本マニュアルでは詳細に説明しません。

ビジュアルプロセスの構成

ビジュアル関数グループ右側の「ビジュアル構成」ボタンをクリックすると、カメラの構成を開始します。初めてカメラプロセスを構成する場合、まず「新規作成」ボタンをクリックし、カメラの名称を入力後、カメラの構成を新規作成し、それから以下の画面のとおりソフトウェアに表示されます。

トリガー方式

カメラをトリガーする方式を設定します。

- I/O内トリガー：カメラとロボットのDOポートを接続するには、電気の配線ポートに基づき対応する出力ポートを構成する必要があります。

- ネットワークトリガー：カメラとロボットのイーサネットポートを接続するには、ネットワークトリガーのための文字列を構成する必要があります。カメラ側とまず事前に合意する必要があり、ロボットはネットワークを介してこの文字列をカメラに送信するとカメラをトリガーします。

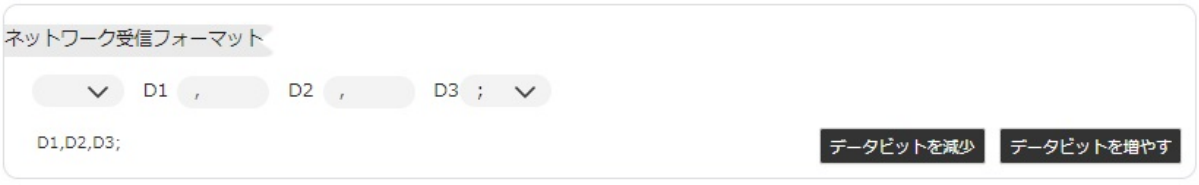
ネットワーク基本パラメータ

基本パラメータは、以下の数種類のモードを含め、カメラとロボットの通信モードを設定するために使用されます。

- TCP_Client：TCP通信で、ロボットをクライアントとし、カメラをサーバーとします。カメラのIP、ポートと接続タイムアウト時間を構成する必要があります。
- TCP_Server：TCP通信で、ロボットをサーバーとし、カメラをクライアントとします。ポートと接続タイムアウト時間を構成する必要があります。

受信方法はブロックとノーブロックの2種類を含みます。プロジェクトのスクリプト設計に基づき選択してください。

- ブロック：トリガー信号を送信した後、プログラムはブロック時間内にデータを受信する行の関数にずっと留まり、カメラから送信されたデータを受信するかブロック時間が超えなければ引き続き実行できません。ブロック時間を0に設定した場合、ビジュアルによって送信されたデータを受信するまでプログラムはデータ受信関数の行にずっと留まります。
- ノーブロック：トリガー信号を送信後、カメラから送信されたデータを受信したか否かにかかわらず、プログラムは継続して実行します。



ネットワークが受け入れる形式は、解析に用いるためカメラから送信されたデータの形式を表します。現在、デフォルトのデータビットで不十分の場合、「データビットを追加」方法で受信データの長さを最大8ビットまで追加することができます。NO、D1、D2、D3、D4、D5、D6、STAで、そのうち、NOは開始ビットのテンプレート番号を表し、STAは終了ビット（ステータスビット）を表します。

複数のビジュアルデータ形式を設定できます。例えば、

- 開始ビットと終了ビットがない：XX、YY、CC
- 開始ビットがあるが、終了ビットがない：NO、XX、YY、CC

- 開始ビットがないが、終了ビットがある：XX、YY、CC、STA
- 開始ビットと終了ビットがある：NO、XX、YY、CC、STA

構成が完了後、右上の「保存」をクリックします。

InitCam

プロトタイプ：

```
InitCam(CAM)
```

説明：

指定のカメラと接続を確立し、初期化します。

必須パラメータ：

CAM：カメラの名称であり、ビジュアルプロセスを構成した時に構成したカメラの名称と一致する必要があります。

戻る：

初期化の結果。

- 0：初期化成功
- 1：初期化失敗

例：

```
InitCam("CAM0")
```

名称がCAM0のカメラを接続および初期化します。

TriggerCam

プロトタイプ：

```
TriggerCam(CAM)
```

説明：

既に初期化されたカメラの撮影をトリガーします。

必須パラメータ：

CAM：カメラの名称であり、ビジュアルプロセスを構成した時に構成したカメラの名称と一致する必要があります。

戻る：

トリガーの結果。

- 0：トリガー成功
- 1：トリガー失敗

例：

```
TriggerCam("CAM0")
```

名称がCAM0のカメラの撮影をトリガーします。

SendCam

プロトタイプ：

```
SendCam(CAM,data)
```

説明：

既に初期化されたカメラにデータを送信します。

必須パラメータ：

- CAM：カメラの名称であり、ビジュアルプロセスを構成した時に構成したカメラの名称と一致する必要があります。
- data：送信するデータ

戻る：

送信結果。

- 0：送信成功
- 1：送信失敗

例：

```
SendCam("CAM0","0,0,0,0")
```

名称がCAM0のカメラにデータを送信し、データの内容は「0,0,0,0」です。

RecvCam

プロトタイプ：

```
RecvCam(CAM,type)
```

説明：

既に初期化されたカメラから送信されたデータを受信します。

必須パラメータ：

CAM：カメラの名称であり、ビジュアルプロセスを構成した時に構成したカメラの名称と一致する必要があります。

オプションパラメータ：

type：受信するデータのタイプ。値の範囲：numberまたはstringで、設定しない時はデフォルトでnumberです。

戻る：

- err：エラーコード
  - 0：正常に受信
  - 1：受信のタイムアウト
  - 2：データ形式の構成エラー、解析できない
  - 3：ネットワークの遮断
- n：データのグループ数。
- data：受信したデータで、二次元アレーに保存します。

例：

```
local err,n,data = RecvCam("CAM0","number")
```

名称がCAM0のカメラから送信されたデータを受信し、データのタイプはnumberです。

DestroyCam

プロトタイプ：

```
DestroyCam(CAM)
```

説明：

カメラとの接続を切断します。

必須パラメータ：

CAM：カメラの名称であり、ビジュアルプロセスを構成した時に構成したカメラの名称と一致する必要があります。

戻る：

切断の結果。

- 0：切断成功
- 1：切断失敗

例：

```
DestroyCam("CAM0")
```

名称がCAM0のカメラとの接続を切断します。

応用事例

カメラのパラメータを構成した後、ビジュアルAPIを呼び出してプログラミングを行うことにより、カメラをトリガーした後にカメラデータの受信を実現します。本マニュアルはスクリプトプログラミングを例としており、CAM0のデータを取得し、ティーチングポイントP2に割り当てるスクリプトを作成します。

```
while true do
   ::create_camera::
    resultInit = InitCam("CAM0")                                    --Connect CAM0 camera
    if resultInit ~= 0 then
        print("Connect camera failed, code:", resultInit)
        Sleep(1000)
        goto create_camera
    end
    while true do
        TriggerCam("CAM0")                                          --Trigger CAM0 camera  photo
        SendCam("CAM0","1,2,3,0;")                                  --Send data to CAM0 camera
        err,visionNum,visionData  = RecvCam("CAM0","number")        --Receive CAM0 camera data
        if err ~= 0 then
            print("Failed to read data")
            Sleep(1000)
            break
        end

        print("(visionNum):",(visionNum))                          --Print how many sets of CAM0 camera data received
        print("(visionData[1][1]):",(visionData[1][1]))            --Print the first data of the first group received
        i = 1
        while not ((visionNum)<i) do
        print(type(P2.coordinate[1]))
        print(P2)
        P2.coordinate[1]=(visionData[i][1])                         --visionData[i][1] is assigned to P2.X
        P2.coordinate[2]=(visionData[i][2])                        --visionData[i][2] is assigned to P2.Y
        Go(P2,"SYNC=1")
        i = i + 1
        end
        Sleep(10)
    end
    Sleep(10)
end
```

コンベアベルト

コンベアベルト命令は、コンベアベルトプロセスパッケージとともに使用する必要があります。

CnvVison

プロトタイプ：

```
CnvVison(CnvID)
```

説明：

指定されたコンベヤベルトを初期化します。

必須パラメータ：

CnvID：コンベアベルト番号。

戻る

結果を設定します。

- 0：成功
- 1：失敗

例：

```
CnvVison(0)
```

番号が0であるコンベアベルトを初期化します。

GetCnvObject

プロトタイプ：

```
GetCnvObject(CnvID, ObjID)
```

説明：

指定されたベルトコンベアから指定されたワークのデータを取得し、掴み領域に入ったかどうかを検出します。

必須パラメータ：

- CnvID：コンベアベルト番号。

- ObjID：ワーク番号。

戻る

- flag：ワークが検出されたかどうか。値の範囲：trueまたはfalse
- typeObject：ワーク分類属性
- point：ワーク座標 {x, y, z}

例：

```
flag,typeObject,point = GetCnvObject(0,0)
```

ベルトコンベア0上のワーク0のデータを取得します。

SetCnvPointOffset

プロトタイプ：

```
SetCnvPointOffset(xOffset,yOffset)
```

説明：

コンベアベルトのユーザー座標系でのX方向とY方向の点のオフセットを設定します。

必須パラメータ：

- xOffset：X軸方向のオフセット、単位：ミリメートル
- yOffset：Y軸方向のオフセット、単位：ミリメートル

戻る

結果を設定します。

- 0：成功
- 1：失敗

例：

```
SetCnvPointOffset(10,10)
```

コンベアベルトの点のオフセットをX=10mm、Y=10mmに設定します。

SetCnvTimeCompensation

プロトタイプ：

```
SetCnvTimeCompensation(time)
```

説明：

時間補正を設定します。ロボットアームの把握位置が実際のワークの位置より遅れている場合、時間補正を設定することで調整することができます。

必須パラメータ：

time：時間偏差、単位：ミリ秒

戻る

結果を設定します。

- 0：成功
- 1：失敗

例：

```
SetCnvTimeCompensation(30)
```

補正時間を30msに設定します。

SyncCnv

プロトタイプ：

```
SyncCnv(CnvID)
```

説明：

指定されたコンベヤベルトを同期します。同期後、コンベアのユーザー座標系に基づいた点がコンベアの動きに追従します。この命令とStopSyncCnv命令の間の移動命令はMove命令のみをサポートします。

必須パラメータ：

CnvID：コンベアベルト番号

戻る

同期結果を返します。

- 0：成功
- 1：失敗

例：

```
SyncCnv(0)
```

番号が0であるコンベアベルトを同期します。

StopSyncCnv

プロトタイプ：

```
StopSyncCnv(CnvID)
```

説明：

同期ベルトコンベアを停止し、この命令を実行した後、引き続き他の命令を発行します

必須パラメータ：

CnvID：コンベアベルト番号

返す

実行結果。

- 0：成功
- 1：失敗

例：

```
StopSyncCnv(0)
```

番号が0であるコンベアベルトの同期を停止します。

セーフスキン

SetSafeSkin

プロトタイプ：

```
SetSafeSkin(status)
```

説明：

セーフスキンスイッチ。

必須パラメータ：

status：0はセーフスキンがオフであることを意味し、1はセーフスキンがオンであることを意味します。Dobot+のセーフスキンがオフになっている場合、この命令は効きません。

例：

```
EnableSafeSkin(1)
```

セーフスキンをオンにします。