

ミスミ QDXP アクチュエータ 取扱マニュアル

株式会社ミスミ



目次

注意事項.....	4
製品紹介.....	4
免責事項.....	4
履歴.....	5
1. 製品情報	6
1.1. モーター主要性能	6
1.2. モーター外観とコネクタおよびインジケータ.....	6
2. MISUMI QDXP Motor Tool	8
2.1. QDXP-USB の準備と接続.....	8
2.2. QDXP-USB 外観	8
2.3. 接続	10
3. MISUMI QDXP Motor Tool ソフトウェアガイド.....	10
3.1. インストールと起動	10
3.2. MISUMI QDXP Motor Tool インターフェース.....	11
3.3. 起動と接続	12
3.4. CONFIGURATION（設定）	12
3.4.1. BASIC（基本設定）タブ	12
3.4.2. Advanced（詳細設定）タブ	14
3.4.3. ファームウェアアップグレード.....	15
3.4.4. リアルタイムステータス.....	15
3.4.5. リアルタイムデータ表示（グラフ画面）	17
3.5. モーター制御.....	18
3.5.1. サーボ制御（Servo Control）モード.....	19
3.5.2. 力制御（MIT）モード	20
3.5.3. 単位設定.....	20
3.6. 接続	21
3.7. 停止	21
3.8. ドライバー基板校正.....	21
3.8.1. 校正手順.....	21
3.9. 制御デモンストレーション	23
3.9.1. Multi-Position-Velocity Loop Mode（多回転位置・速度ループモード）	23
3.9.2. 単回転位置・速度ループモード.....	23
3.9.3. ブレーキモード	24
3.9.4. Position Loop（位置フィードバック）モード	25
3.9.5. Current Loop（電流フィードバック）モード	25
3.9.6. Velocity Loop（速度フィードバック）モード.....	26
3.9.7. Braking Loop（ブレーキ）モード.....	26
3.9.8. Duty Cycle（デューティ比入力）モード.....	27

3.9.9.	Force Control（力制御・MIT）モード.....	27
3.10.	ファームウェアアップグレード.....	28
4.	ドライバー基板通信プロトコル.....	30
4.1.	サーボ制御モード.....	30
4.1.1.	デューティサイクルモード（Duty Cycle Mode）.....	32
4.1.2.	電流ループモード.....	33
4.1.3.	電流ブレーキモード.....	33
4.1.4.	速度ループモード.....	34
4.1.5.	位置ループ（位置フィードバック）モード.....	35
4.1.6.	原点設定モード.....	35
4.1.7.	位置・速度ループモード.....	36
4.1.8.	モーター無効化（モーターOFF）モード.....	37
4.1.9.	フィードバックメッセージ設定.....	37
4.2.	力制御モード（MIT モード）通信プロトコル.....	38
4.3.	モーターCAN メッセージフォーマット.....	43
4.3.1.	CAN 送信（ホスト→モーター）メッセージプロトコル.....	43
4.3.2.	シリアルポートメッセージプロトコル.....	45
4.3.3.	パラメータの取得.....	45
4.3.4.	制御コマンド 1.....	48
4.3.5.	シリアルポートキャリブレーション（serial port calibration）.....	52
4.4.	制御コマンド例.....	53
4.4.1.	CAN ポートを使った制御コマンド例.....	53
4.4.2.	シリアルポート制御コマンド例.....	54

注意事項

1. 配線経路など回路内に電気の短絡が無いこと、電源・CAN 通信ラインなどの各インターフェースが本マニュアル通りに正しく接続されていることを確認してください。
2. モーター内部のドライバー基板は動作中に発熱します。やけどを避けるため、取り扱いには十分注意してください。
3. 使用前にすべての部品が損傷なく正常であるか確認してください。部品が欠品している、または劣化している場合は使用を中止し、速やかにミスミお客様サービス <https://jp.misumi-ec.com/inquiry> へ連絡してください。
4. 本マニュアルに記載された動作電圧・電流・温度などの設定値（以下パラメータと記述します）を厳守してください。違った使い方をすると製品に永久的な損傷を与えるおそれがあります。

製品紹介

ミスミ QDXP モーターシリーズは同クラスの高性能ドライブチップを採用し、FOC（ベクトル制御）アルゴリズムにより外乱に強い速度制御および角度制御のための高度な制御技術を備えています。

パラメータ設定ソフトウェア MISUMI QDXP Motor Tool と併用することで、パラメータの調整やファームウェアアップグレードを行うことができます。

ハードウェア面では内部ループに 16 ビット高精度エンコーダを使用し、最大 21 ビットまで対応します。また、CAN 通信にはより安全な絶縁 CAN インターフェース（Isolate CAN）を採用し、より信頼性の高いプラグと組み合わせることで製品の使用および通信の信頼性を大幅に向上させています。

ソフトウェア面では、上位 PC 用 MISUMI QDXP Motor Tool によりサーボモードと MIT モードを切り替える必要がなくなり、制御インターフェースがより簡潔になりました。また操作も大幅に簡素化され、ユーザーエクスペリエンスの高いソフトウェアになっています。

免責事項

ミスミ QDXP モーターシリーズをご購入いただき誠にありがとうございます。ご使用前に本免責事項をよくお読みください。使用した時点で免責事項のすべての内容を承認し受け入れたものとみなされます。製品の取り付けおよび使用にあたっては、製品マニュアルならびに関連する法律、規制、方針、ガイドラインを厳守してください。製品の使用中、ユーザーは自身の行為およびそれにより生じるすべての結果について責任を負うことを約束するものとします。

ユーザーによる製品の不適切な使用、取り付け、または改造によって生じたいかなる損失についても、株式会社ミスミは法的責任を負いません。

ミスミ QDXP モーターは株式会社ミスミおよびその関連会社の商標です。本書に記載されている製品名およびブランドは各社の商標です。

本製品および本マニュアルの著作権は、株式会社ミスミに帰属します。許可なく複製または転載することはできません。免責事項の最終的な解釈権は株式会社ミスミに帰属します。

日付	2026/08/04	Version 1.0	基板ドライバーバージョン 3.0
----	------------	-------------	------------------

1. 製品情報

1.1. モーター主要性能

ミズミ QDXP モーターシリーズの主要性能は下記のようになっています（2026 年 8 月 27 日）

型番	外径	高さ	重量	減速比	定格電圧	定格トルク	ピークトルク	定格出力	無負荷
	(mm)	(mm)	(g)		(V)	(Nm)	(Nm)	(W)	回転数(rpm)
QDXP1-53-2.5-24-C	53	45.2	262	10:1	24	2.5	7	39	180
QDXP1-53-2.5-48-C	53	45.2	262	10:1	48	2.5	7	39	180
QDXP2-79-3-W-C	79	43	380	6:1	24/48	3	9	74	320/640
QDXP2-79-24-48-C	79	67	750	39:1	48	24	72	176	98
QDXP2-89-8.5-48-C	89	49	540	9:1	48	8.5	29.2	231	320
QDXP2-98-9-48-C	98	38.5	490	9:1	48	9	22	245	570
QDXP3-98-18-48-C	98	61.7	940	9:1	48	18	53	217	320
QDXP4-90-26-48-C	90	60.5	879	16:1	48	26	78	442	105
QDXP4-90-74-48-C	90	81.5	1396	48:1	48	74	222	368	35

1.2. モーター外観とコネクタおよびインジケータ

モーターとのケーブル接続について「QDXP2-79-24-48-C」を例にとり説明します。

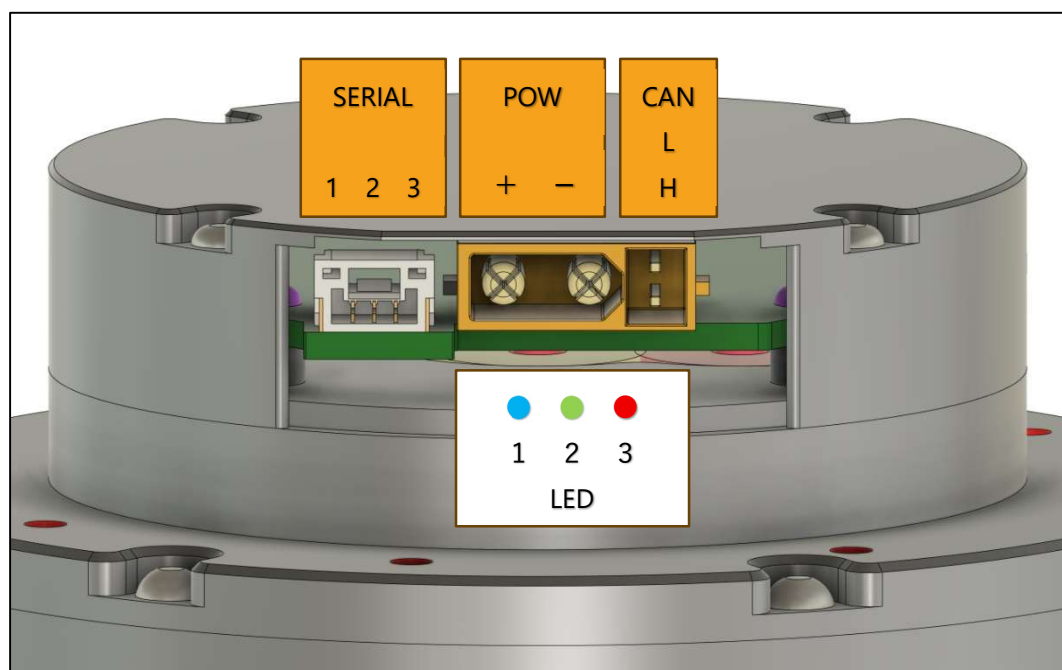


図 1 QDXP2-79-24-48-C コネクタ取り付け部

表 1 モータードライバー基板コネクタピンアサイン

No	インターフェース 機能	Pin	内容	ケーブルカラー *1
1	電源入力	1	電源 (+)	Red
		2	電源 (-) グランド	Black
2	CAN通信	1	CAN通信ライン L (CAN_L)	Blue
		2	CAN通信ライン H (CAN_H)	White
3	シリアル通信	1	グランド (GND)	Black
		2	シリアル入力 (RX) *2	Yellow
		3	シリアル出力 (TX) *3	Green

*1 モーター付属ケーブルのカラーです。

*2 モーター側入力 (RX) は QDXP-USB では出力 (TX) になります。

*3 モーター側出力 (TX) は QDXP-USB では入力 (RX) になります。

表 2 使用コネクタ型番

No	機能	基板コネクタ 型番	メーカー	ケーブル端末	メーカー
1	シリアル通信	A1257WR-S-3P	CJT	A1257H-3P	CJT
				GHR-03V-S	日本圧着端子*
2	電源・CAN通信	XT30PW(2+2)-M	AMASS	XT30(2+2)-F	AMASS

* GHR-03-S で代用可能

他のモーターもピンアサインは同じです。

表 3 インジケータ点灯状態

No	インジケータ点灯状態		
1	電源インジケータ (青)	点灯	ドライバー基板に電源が供給されています
		消灯	ドライバー基板に電源が供給されていません
2	操作インジケータ (緑)	点灯	モーターは動作中です
		消灯	モーターは動作していません
3	ドライバー基板エラー (赤)	点灯	ドライバー基板にエラーが発生しています
		消灯	ドライバー基板は正常です

2. MISUMI QDXP Motor Tool

MISUMI QDXP Motor Tool はモーターのパラメータを設定したり、グラフによって動作確認ができる Windows 用ソフトウェアです。

2.1. QDXP-USB の準備と接続

- Windows PC（Windows10 または Windows11）、仮想 COM ポートドライバ* が動作すること。
* 標準で Windows にインストールされていますが、動作しない場合は CH343 VCP ドライバをインストールする必要があります。
 - 接続用 USB Type A ポート
 - USB-シリアル変換アダプター MISUMI QDXP-USB
 - ミスミ QDXP Motor Tool
 - モーターに対応する 24V または 48V 電源（対応するバッテリーまたは安定化電源）
 - モーターとモーター付属ケーブル
- モーターと PC は 1 対 1 で接続します。

2.2. QDXP-USB 外観

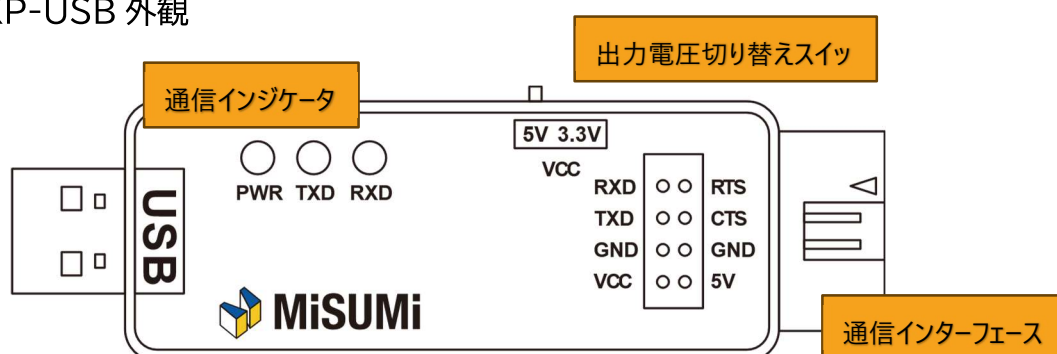


図 2 QDXP-USB 外観

表 4 QDXP-USB 電氣的仕様

項目	説明
動作電圧	5V（USBから供給）
電圧切り替えスイッチ	5.0V または 3.3V(標準)
待機消費電流	≤30mA
大きさ	73.8 x 23.6 x 14.5 mm
動作環境温度	-20℃ から 65℃ まで
許容最大温度	85℃

表 5 通信インターフェース（ピンヘッダ）ピンアサイン

番号	機能	ピン番号	説明
1	シリアルインターフェース	1	シリアル信号入力（RXD）
		2	シリアル信号出力（TXD）
		3	シリアル信号グラウンド（GND）
		4	供給電圧（VCC）＊
		5	データ送信要求（RTS）
		6	送信許可（CTS）
		7	電源グラウンド（GND）
		8	5V出力（5V）

＊ 電圧切り替えスイッチ設定電圧が出力されます。

表 6 通信インジケータ点灯ステータス

番号	カラー	説明
1	Red	PWR電源インジケータはQDXP-USBの電源状態を示します。通常は電源が接続されると赤色に点灯します。 電源を接続しても点灯しない場合は直ちに電源を取り外して、再度電源を入れようとせずにサービスにお問い合わせください。
2	Red	TXDシリアル通信出力（TX）は通常消灯しており、QDXP-USBのシリアルポートからデータが出力されると点滅します。
3	Red	RXDシリアル通信入力（RX）は通常消灯しており、QDXP-USBのシリアルポートへデータが入力されると点滅します。

2.3. 接続

モーターと電源、QDXP-USB との接続は下記のようにモーター付属ケーブルを使用します。モーター付属電源・CAN ケーブルのコネクタに取り付けられている CAN ケーブル（白・青）は MISUMI QDXP Motor Tool では使用しません、他の端子などとショートしないように注意してください。

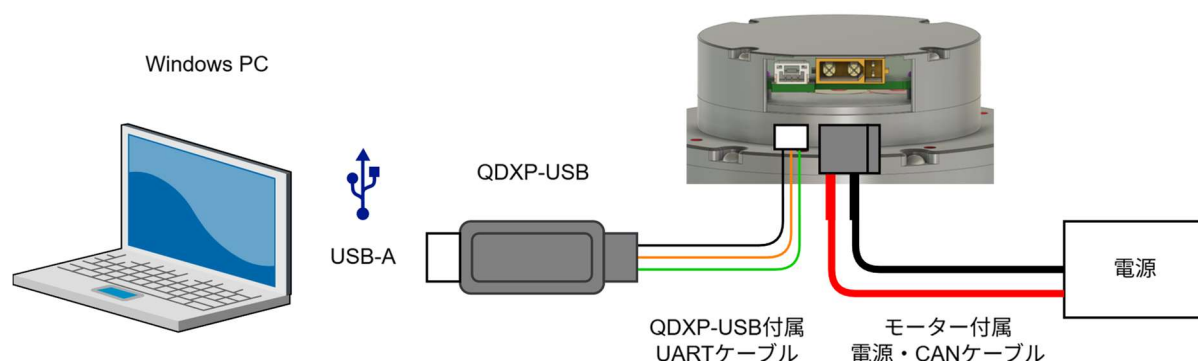


図 3 モーターと QDXP USB の接続

PC の USB-A ポートに QDXP-USB を接続し、デバイスマネージャの「ポート（COM と LPT）」を開き、「USB-Enhanced-SERIAL CH343（COM?）」*といった仮想 COM ポートを認識しているかご確認ください（図 4 デバイスの認識を参照）。Windows 標準では CH343 仮想 COM ポートドライバをインストールする必要はありませんが、デバイスマネージャで警告マークなどが表示されデバイスを認識していない場合はドライバをインストールしてください。

*「?」には数値が表示されます。図 4 では COM5 という名前で仮想シリアルポートが認識されています。

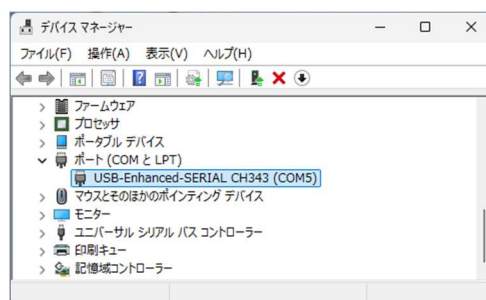


図 4 デバイスの認識

3. MISUMI QDXP Motor Tool ソフトウェアガイド

3.1. インストールと起動

インストールはありません。ダウンロードした ZIP 形式ファイルを PC の適当なフォルダに解凍して、MISUMI QDXP Motor Tool.exe をダブルクリックして起動してください。

起動後に下記のようなダイアログが表示された場合は、「詳細情報」をクリックすると「実行」ボタンが表示されますので、「実行」ボタンを押して MISUMI QDXP Motor Tool を起動してください。



ご使用のプロテクトソフトウェアにより違う画面が出る可能性があります。

3.2. MISUMI QDXP Motor Tool インターフェース

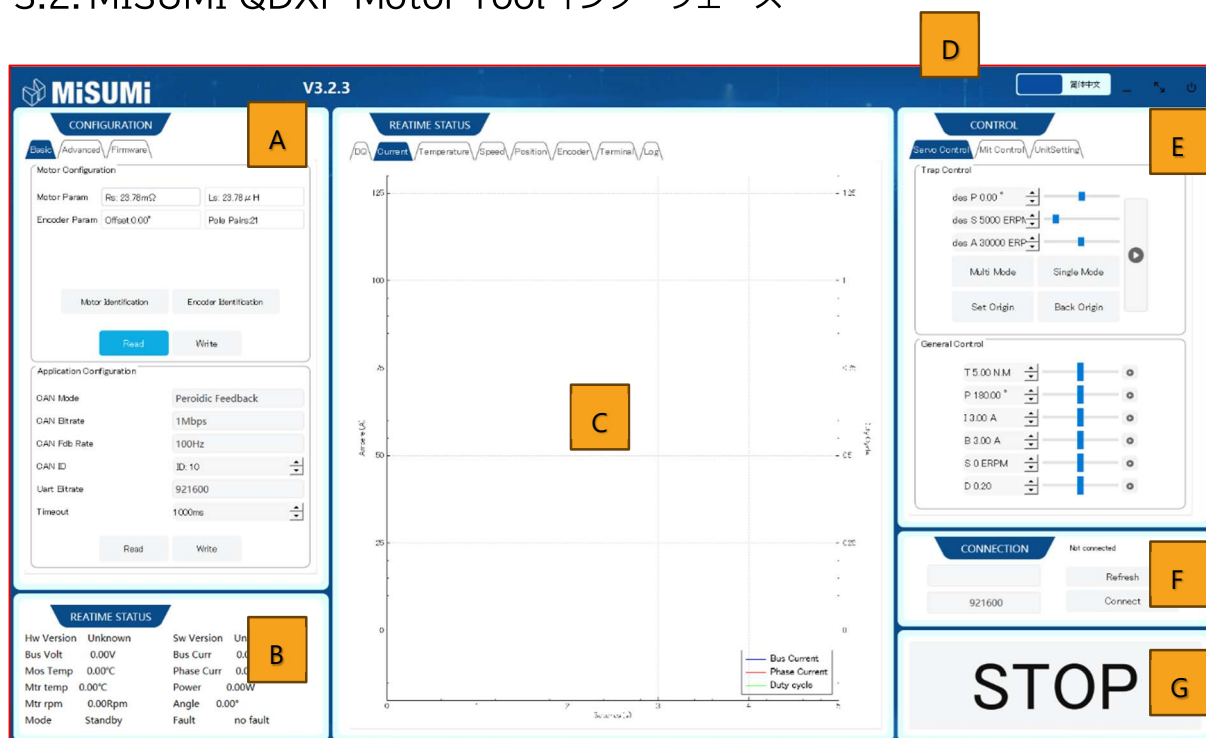


図 5 MISUMI QDXP Motor Tool 起動時画面（英語表記時）

A. CONFIGURATION :

モーターパラメータ設定タブです。CONFIGURATION タブには BASIC、Advanced、Firmware タブがあります。

B. REALTIME STATUS :

モーターのリアルタイムステータス表示エリア（入力電圧・モーター温度など）

C. REAL TIME DATA :

モーターのリアルタイムデータの表示エリア（角度・速度・電流など）

D. Chinese-English Switch :

英語・中国語切り替えスイッチ

E. CONTROL :

モーターへの指令値（コマンド）入力エリア

F. CONNECTION :

仮想 USB ポートとの接続と設定エリア

G. STOP :

モーター停止ボタン

起動時は中国語表記になっていることがあります。MISUMI QDXP Motor Tool のタイトルバー右上（図 5 「E」）にある「中」または「EN」をクリックし、中国語または英語に表示言語を切り替えてご利用ください。

3.3. 起動と接続

MISUMI QDXP Motor Tool を起動したら、「F. Connection」セクションで QDXP-USB の仮想 COM ポートを接続してください。詳しくは「3.6 接続」をお読みください。

3.4. CONFIGURATION(設定)

3.4.1. BASIC(基本設定)タブ

Motor Configuration セクション

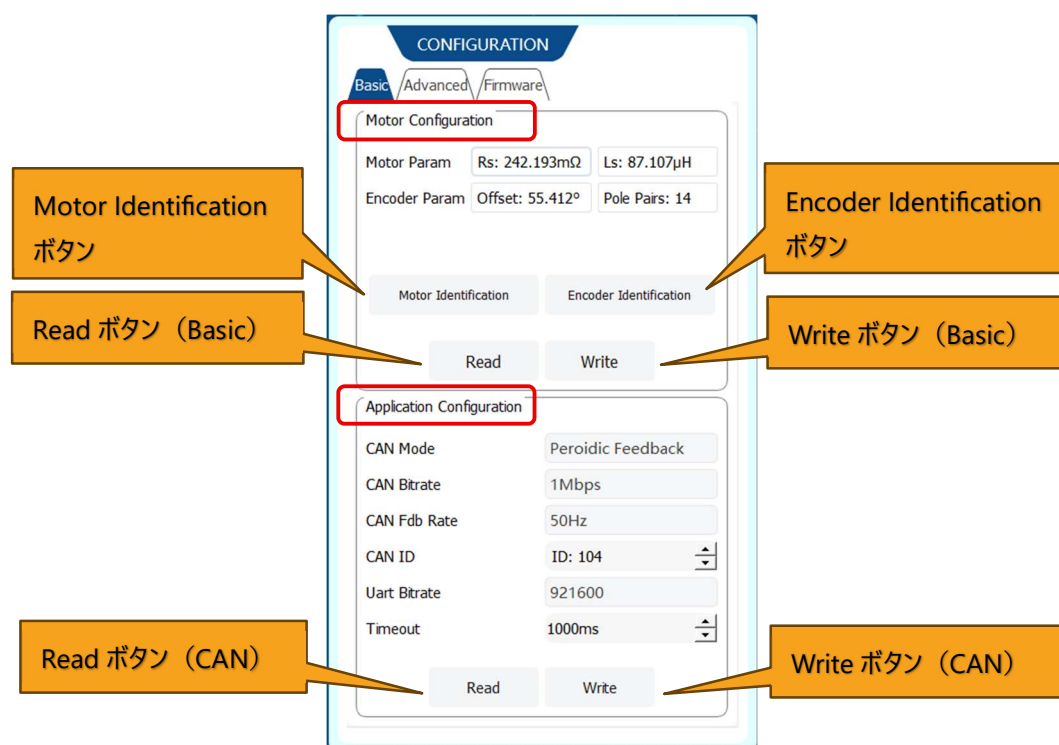


図 6 CONFIGURATION・BASIC タブ

BASIC タブではユーザーがモーターパラメータおよび CAN 通信パラメータを設定できます。基本設定画面はいくつかのセクションに分かれていて、モーター設定セクションではモーターパラメータの識別、エンコーダパラメータの校正、パラメータの

読み取りおよび書き込みができます。具体的なモーターおよびエンコーダ校正については、3.8 節を参照してください。
アプリケーション設定（Application Configuration）セクションでは CAN 通信モード、CAN ビットレート、CAN フィードバックレート、CAN ID、およびシリアルポートボーレートの設定ができます。

Motor Identification ボタン：

モーターのパラメータを確認します。グラフの Position（角度）でモーターが動いているかどうかを確認できます。確認が完了すると、自動的に停止し、モーターのパラメータがソフトウェア画面に更新されます。

△：モーターパラメータ確認の全工程は無負荷状態で行ってください。そうしない場合、識別パラメータが不正確になったり、モーターが損傷したりするおそれがあります。識別プロセスは鋭く短い音から始まり、その後モーターが大きめの音を伴って回転し始めます。

Encoder Identification ボタン：

エンコーダパラメータ識別を実行します。位置曲線でモーターが動作しているかを確認できます。

Read ボタン（Basic）：

モーターのドライバー基板からモーターパラメータおよびエンコーダパラメータを読み出します。

Write ボタン（Basic）：

ソフトウェアで設定したモーターパラメータおよびエンコーダパラメータをモーターのドライバー基板へ書き込みます。

Application Configuration セクション

Application Configuration セクションではモーターの CAN 通信に関するパラメータの設定ができます。

CAN Mode（CAN 通信モード）： Periodic Feedback または Inquiry-Reply

- Periodic Feedback（周期フィードバック）モードはモーターがステータスを定期的にホスト側に送るモードです。周期は「CAN フィードバック周期」欄で変更します。
- Inquiry-Reply（問い合わせ返信）モードはホスト側からのコマンド送信に対してモーターが返事するモードです。

CAN Btrrate（CAN 通信ビットレート）：

CAN 通信のビットレートを決めます。デフォルトで 1Mbps（1 Megabit per second）を選択します。

CAN Fdb Rate（CAN フィードバック周期）：

モーターからホストへのデータ送信周期(単位：Hz = (1/Hz)秒)

CAN ID：

CAN 通信では各モーターおよびホスト PC に ID 番号が必要です。モーターに固有の番号をつける際は CAN ID 欄を使用します。

Uart Btrrate（UART 通信ビットレート）：

QDXP-USB とモーター間の通信ビットレートの設定です。特に変更する必要はありません。

Read ボタン（CAN）：

Application Configuration のパラメータをモーターから読み取ります。

Write ボタン（CAN）：

Application Configuration のパラメータをモーターへ書き込みます。

3.4.2. Advanced(詳細設定)タブ

Advanced タブを表示したときは図 7 のような警告が表示されます。



図 7 Advanced タブ表示時警告

Advanced タブではモータードライバー基板のモーター制御に密接に関連したパラメータが表示されるため、変更すると重大な問題を引き起こす可能性がありますので注意してください。

△警告：Advanced タブに表示されるパラメータを変更しないでください。変更するとモーターが異常動作する可能性があります。

Advanced Motor Configuration セクション

Advanced タブでは、ユーザーが必要に応じてより多くのパラメータをカスタマイズできます。パラメータの読み取りおよび書き込み機能に加えて、パラメータの復元、読み込み、エクスポート機能も提供しています。

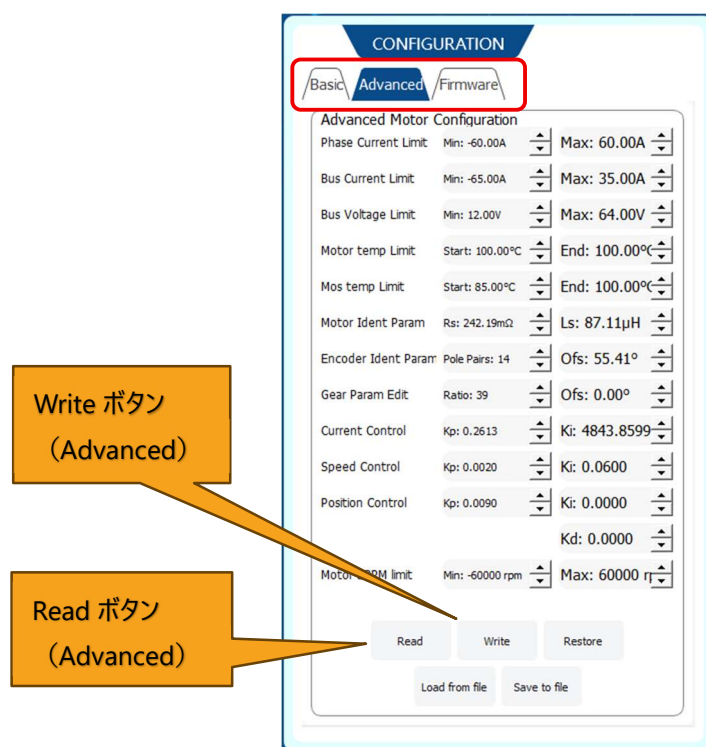


図 8 CONFIGURATION・Advanced タブ

Read ボタン (Advanced) :

クリックすると、ドライバー基板から上位コンピューターへパラメータを読み取ります。

Write ボタン (Advanced) :

クリックすると、現在の上位コンピューター側パラメータをドライバー基板へ書き込みます。

△：モーターパラメータを書き換える前に、必ず最初にパラメータ読み込みを行ってください。そうしない場合、その他のモーターパラメータが正しくなくなります。この状況が発生した場合は、ミスミサポート窓口へ連絡のうえ、該当モデルパラメ

ータータを送付させていただきます。「Load from file」ボタンを使ってモーターの標準値を書き込んでください。

Restore ボタン（デフォルト値への復元）：

クリックすると、上位コンピューター側のパラメータを対応モデルのデフォルト値に復元します。

Load from file ボタン：パラメータファイルの読み込み

パラメータファイルを MISUMI QDXP Motor Tool に読み込みます。クリックして対応するモーターモデルのパラメータファイルを選択します。対応するモーターモデルのパラメータファイルは、「.AppParams」および「.McParams」の 2 つの形式があります。

Save to file：パラメータのファイルへのエクスポート

クリックして保存先ファイルを選択すると、カスタムパラメータファイルを PC 上のファイルにエクスポートできます。

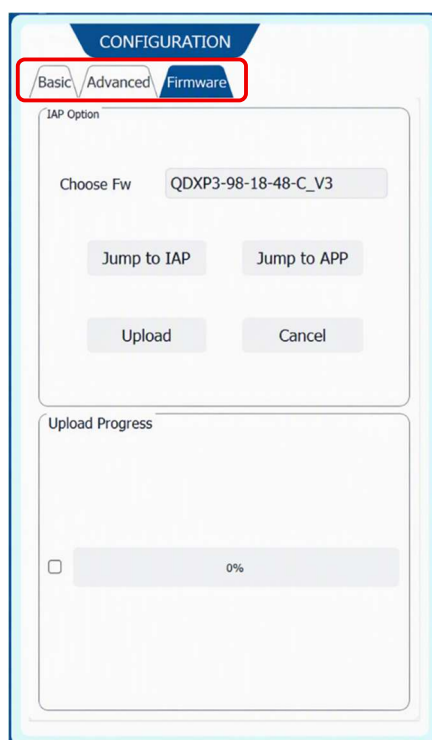


図 9 Firmware タブ

3.4.3. ファームウェアアップグレード

ファームウェアアップグレード画面（タブ）にはファームウェア情報表示、アップグレードオプション、アップグレード進捗の 3 つのセクションがあります。お客様は必要に応じてファームウェアをアップグレードできます。ファームウェアアップグレードの具体的な手順については、3.10 節を参照してください。

3.4.4. リアルタイムステータス

リアルタイムステータス画面ではモーターのソフトウェアおよびハードウェアバージョン、ファームウェアバージョン、モーターステータス、モード、エラー状態を表示します。

REALTIME STATUS			
Hw Version	QDXP3-98	Sw Version	QDXP3-98
Bus Volt	44.60V	Bus Curr	0.00A
Mos Temp	46.60°C	Phase Curr	0.41A
Motor Temp	31.10°C	Power	0.00W
Out rpm	0.00Rpm	Angle	79.79°
Mode	Standby	Fault	no fault

図 10 リアルタイムステータス表示

リアルタイムデータのフィードバックを行い、デバッグ画面とグラフを表示します。データには以下が含まれます。

Bus Volt（バス供給電圧）：

モーターに供給される電源電圧を表示します。

Bus Curr（バス電流）：

電源からモーターへ流れる全ての電流を表示します。

Temperature（温度）：

Mos Temp はモーターの MOS トランジスタの温度を示し、Motor Temp はモーター温度を表示します。

Out rpm（速度）：

モーター出力軸の回転速度（RPM）を表示します。

Angle（角度）

モーター出力軸の回転角度（degree）を表示します。

Mode：

現在のモーター動作モードを表示します。

Fault：

モーターのエラー状態を表示します。通常、問題が無いときは「no fault」と表示されます。

3.4.5. リアルタイムデータ表示(グラフ画面)

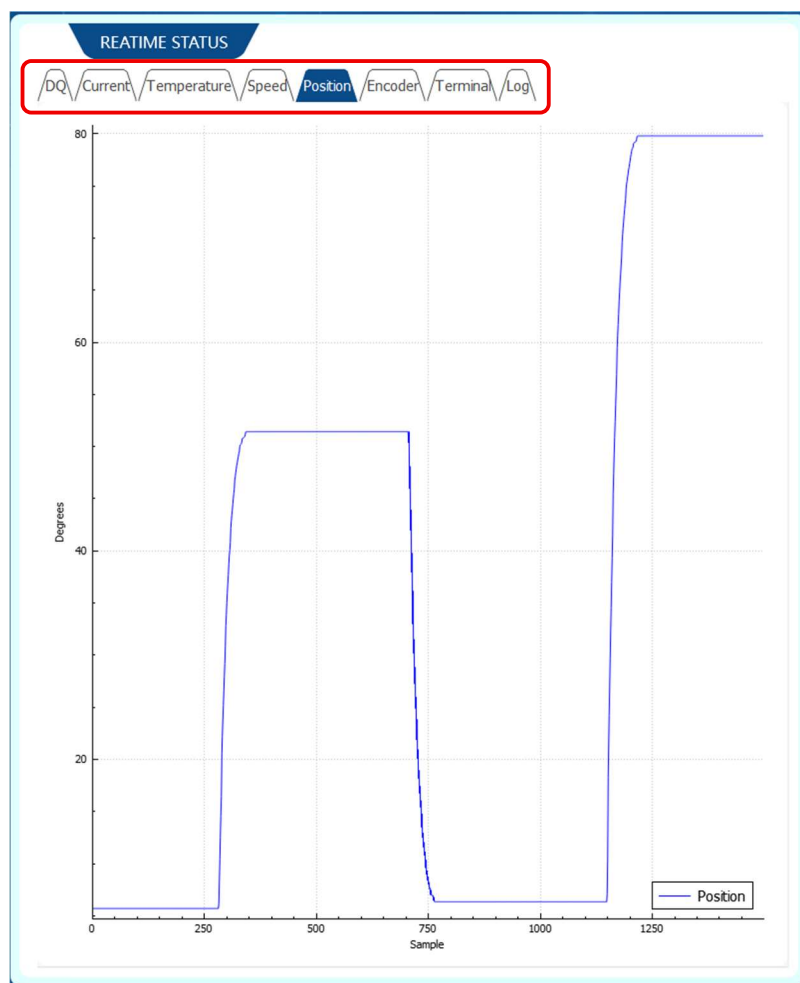


図 11 REALTIME データ表示

DQ タブ：

DQ 電圧はモーター電圧を D 軸 * および Q 軸上に変換した値を示し、DQ 電流はモーター電流を D 軸および Q 軸上に変換した値を示します。

Current タブ：

Phase Current（相電流）：モーター各相の巻線電流のことです。

BUS Current（バス電流）：モーターに供給される全電流です。

Duty cycle（デューティ比）：モーターへ出力される PWM 信号のデューティ比を示しています。

Temperature（温度）：

モータードライバー基板の温度を表示します。

Speed（速度）：

モーターの極の回転速度（ERPM*）を表示します。UnitSetting 設定でグラフの表示単位を変換できます。

Position（位置）：

モーター出力軸の角度を表示します。

Encoder angle（エンコーダ角度）：

エンコーダの角度を表示します。

Terminal (シリアル端末メッセージ出力) :

モーターから送信されるメッセージを表示します。

Log (ログ保存) :

Set Path ボタンでログの保存フォルダを指定すると、モーターの動作データを記録し CSV 形式ファイル (セミコロン「;」区切りファイルになります) に保存します。

* D 軸および Q 軸とは、モーターのベクトル制御において三相電圧または電流を座標変換して得られる 2 軸のことです。D 軸 (Direct Axis) は磁束成分とその方向で、それに直交する電流成分 (トルク成分) が Q 軸 (Quadrature Axis) と呼ばれています。

* ERPM とは通常モーターの 1 極が 1 周回する回転速度です。本 QDXP モーターでは出力回転数 RPM は、ERPM (指令値) に対してモーターの極数とギア比でそれぞれ割った数となります。

Output speed = ERPM (electrical speed) ÷ number of pole pairs ÷ reduction ratio

出力速度 (RPM) = ERPM ÷ 極対数 ÷ ギア比

極対数 = 極数 ÷ 2

例：極対数が 14、ギア比が 39 の QDXP2-79-24-48-C モーターで ERPM を 8000 と設定したときの出力軸速度を求める

$$\begin{aligned}\text{出力速度 (RPM)} &= 8000 \text{ (ERPM)} \div 14 \div 39 \\ &= 14.65 \text{ (RPM)} \\ &= 87.91 \text{ (degree/sec)}\end{aligned}$$

3.5. モーター制御

ミスミ QDXP シリーズモーターにはサーボモードと MIT モード (力制御モード) という 2 つのモーター制御モードがあります。両者は V2.0 までは切り替えて排他的に使用していましたが、V3.0 から混在して使えるようになりました。

3.5.1. サーボ制御(Servo Control)モード

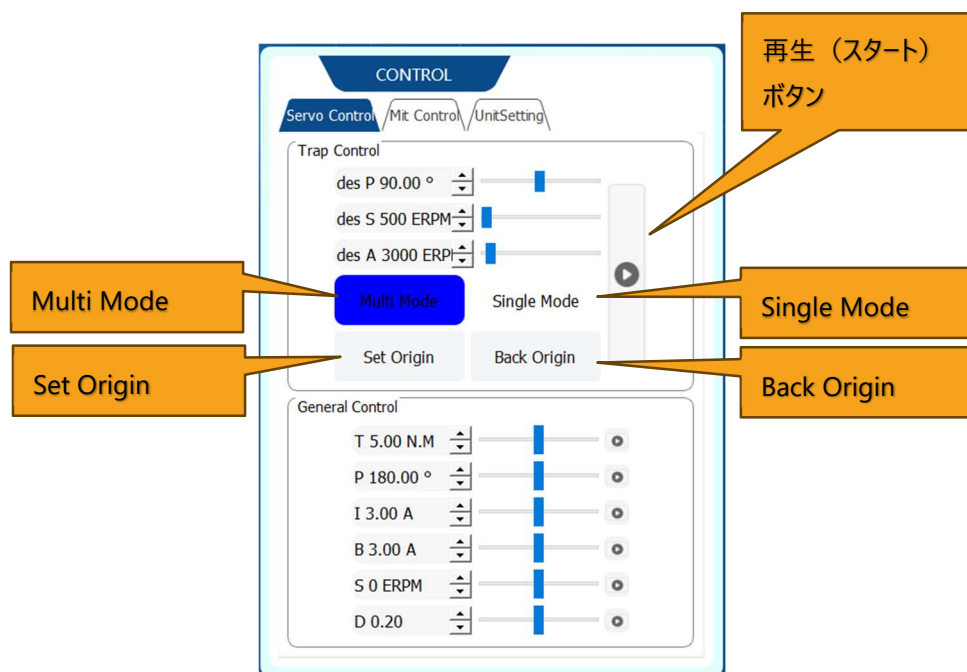


図 12 Servo Control

Trap Control セクション

MULTI Mode ボタン (多回転モード) :

ボタンを押すとモーターを 1 回転以上回転させることができる Multi Mode になります。指定可能な範囲は-36000° ~36000° です。

SINGLE Mode ボタン (単回転モード) :

ボタンを押すとモーターは最大 1 回転まで回転できる Single Mode になります。指定可能な範囲は 0° から 359° です。

Set Origin ボタン (原点合わせ) :

モーターは現在のエンコーダ位置をゼロ位置 (原点) として設定します。

Back Origin ボタン (原点に移動) :

モーターはゼロ位置まで回転します (モーターは最大速度で動作するため、安全に注意してください)

Start (再生ボタン) :

上記 4 つのモードを選択してからこのボタンを押すと動作開始します。

設定できるパラメータは下記のようにになっています。

des P (目標位置) :

モーターの回転角度 (目標位置) を「°」(degree) で指定します。

des S (目標角速度) :

モーターの回転速度を「ERPM」で指定します。

des A (目標加速度・減速度) :

モーターが des P で指定された角度になるために、停止状態から des S になるための加速度を「ERPM/秒」で指定します。

General Control セクション

General Control セクションを利用したモーター動作については「3.9 制御デモンストレーション」を参照してください。

3.5.2. 力制御(MIT)モード

力制御モード（モーション制御モード、MIT MODE）は、位置、速度、トルクループの3種類の制御モードを提供し、送信されるデータによって決定されます。具体的な操作方法については、3.9.9 節を参照してください。

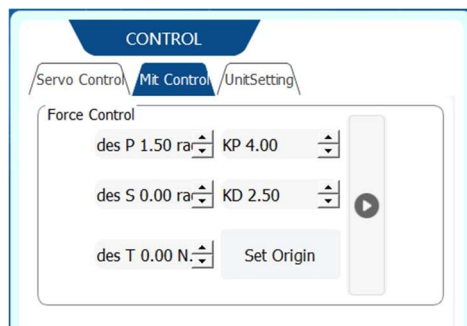


図 13 MIT Control Mode

3.5.3. 単位設定

各ボタンは回転速度（RPM、ERPM、rad/s、Dps）の変換表示に対応しています。

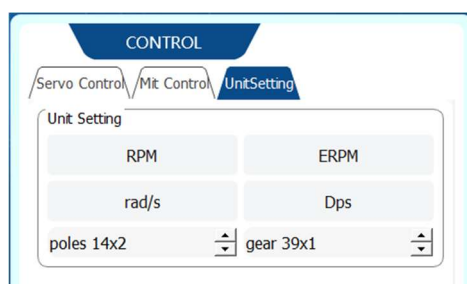


図 14 Unit Setting（単位設定）タブ

RPM : Revolutions per Minute

回転数（角速度）を RPM（毎分あたり回転数）で表示します。

ERPM : Electrical RPM

極対数の毎分あたり回転数に変換します。

rad/s : Radians per Second

回転速度をラジアン毎秒にします。

Dps : Degrees per Second

回転速度を 1 秒あたりの角度（°）にします。

Poles・Gear :

それぞれモーターの極数とギア比です。MISUMI QDXP Motor Tool でモーターに接続すると自動的にモーターの設定値を読み出して表示します。**変更しないでください。**

3.6. 接続

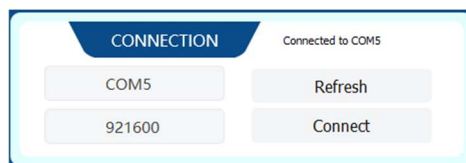


図 15 Connection（接続設定）

接続セクションでは、QDXP-USB のシリアル通信を設定できます。

- 最初に「Refresh」をクリックします。QDXP-USB が PC に接続されていて、仮想 COM ポートが認識されている場合、Refresh ボタンの左側エリアに仮想 COM ポート名が表示されます（「図 15」画像では「COM5」となっています）。
- 通信ボーレートを選択します。通常は 921600bps を選択してください。
- 「Connect」ボタンをクリックします。表示が「Not connected」から「Connected to COM?» * に変わると、シリアル接続が成功したことを示します。変化が無いときは仮想 COM ポートが間違っているか、モーターがつながっていないか、モーターに電源がつながっていないかなどを確認してください。
* ? には数値が入ります。
- 接続が成功すると「Connect」ボタンは「Disconnect」ボタンに変わります。Disconnect ボタンをクリックすると、通信を切断できます。

3.7. 停止

STOP :

動作停止ボタンです。クリックすると、モーターの動作を停止します。

3.8. ドライバー基板校正

モーターにドライバー基板を再取り付けした場合やモーターの三相線の配線順序を変更した場合、またはファームウェアを更新した後は再校正が必要です（初回出荷時は校正済みのため再校正の必要はありません）。校正後にモーターを使用できます。

3.8.1. 校正手順

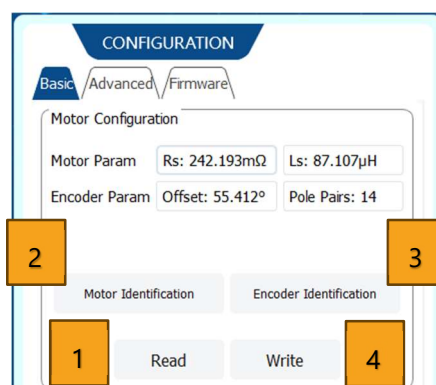


図 16 Calibration Steps（校正手順）

【STEP 0】

モーターに安定的に電気を供給していること、コネクタが正しく接続されていること、PC とモーターが QDXP-USB で正常に接続されていて、MISUMI QDXP Motor Tool でモーターが認識されていることを確認し、MISUMI QDXP Motor Tool のシステム設定ページを表示します。

【STEP 1】

「Read」をクリックし、モーターパラメータが画面に表示されるまで待ちます。

【STEP 2】

「Motor Identification」をクリックしてください。短いピープ音の後、モーターが回転を開始します。約 10 秒待つと、モーターの回転が停止します。

表示が現れたら、モーターパラメータ識別プロセスが完了したことを示します。

【STEP 3】

「Encoder Identification」をクリックすると、モーターがゆっくり回転します。約 45 秒待ち、接続画面の表示が完了すると、エンコーダパラメータ識別が終了します。

【STEP 4】

「Write」をクリックし、接続画面の表示が完了すると、校正は終了です。

△：モーターパラメータおよびエンコーダパラメータ識別の全工程は、無負荷状態で行ってください。そうしない場合、識別パラメータが不正確になったり、モーターが損傷したりするおそれがあります。エンコーダパラメータ識別プロセスでは発熱が発生し、この処理を連続して複数回実行すると、モーター温度が急激に上昇する可能性があります。

3.9. 制御デモンストレーション

本節ではサーボ制御モードおよび力制御モードの使用法について説明します。MISUMI QDXP Motor Tool でモーター動作を実行する際は①モーターに安定的に電気を供給していること、②コネクタが正しく接続されていること、③PCとモーターが QDXP-USB で正常に接続されていることをご確認の上で実行してください。

3.9.1. Multi-Position-Velocity Loop Mode(多回転位置・速度ループモード)

「Servo Control」画面で①「Multi Mode」をクリックし、②目標位置＊、目標速度、加速度を入力します。③スタートボタンを押すとモーターは目標位置に到達するまで、目標速度で移動します。

＊角度指令値の範囲は±100 回転（-36000°～36000°）です。

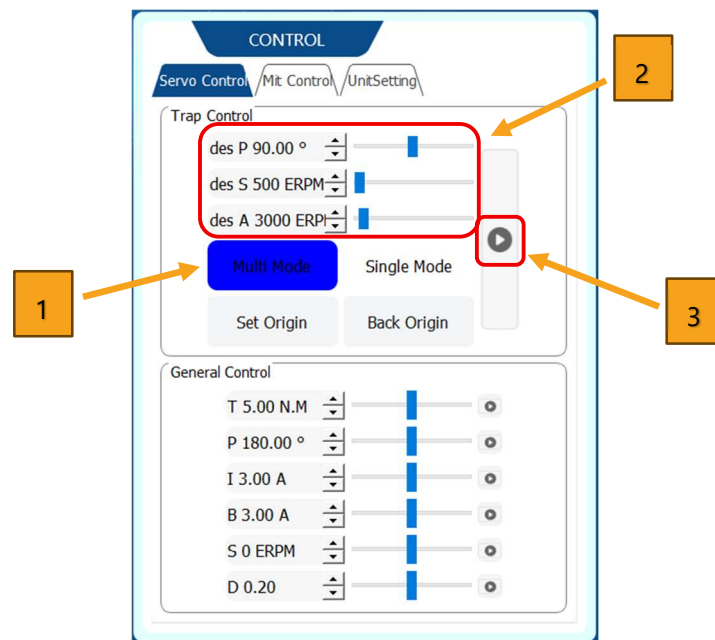


図 17 Multi Mode での位置・速度フィードバック制御

3.9.2. 単回転位置・速度ループモード

「Servo Control」画面で①「Single Mode」をクリックし、②目標位置＊、目標速度、加速度を入力します。③スタートボタンを押すとモーターは目標位置に到達するまで、目標速度で移動します。

＊角度指令値の範囲は1 回転のみ（0～359°）です。

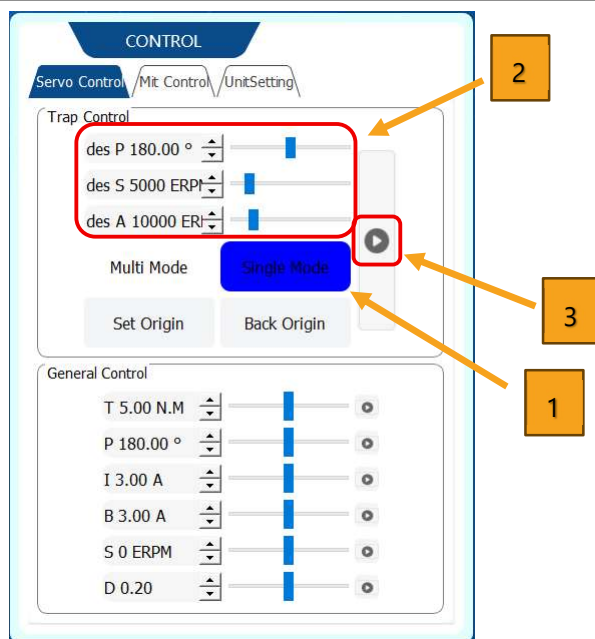


図 18 Single Mode での位置・速度フィードバック制御

3.9.3. ブレーキモード

「Servo Control」タブの「General Control」セクション画面で目標トルク T を入力して右のスタートボタンを押すと、モーターは目標トルクでブレーキをかけます。

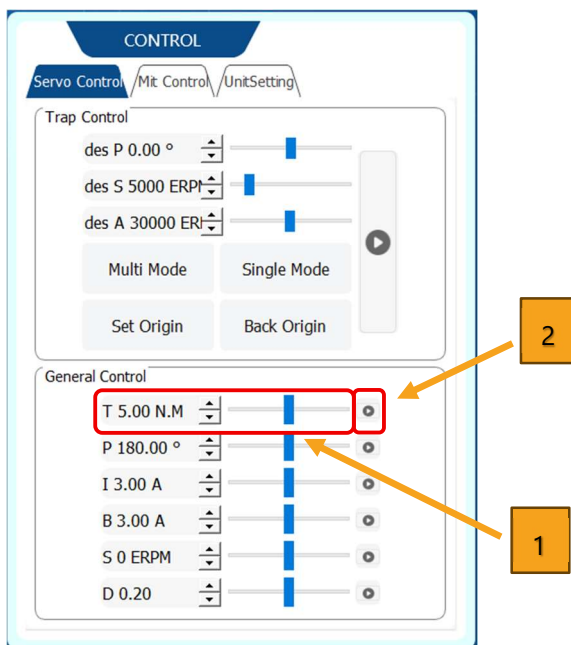


図 19 Braking モード

3.9.4. Position Loop(位置フィードバック)モード

「Servo Control」タブの「General Control」セクション画面で目標位置 P を入力して右のスタートボタンを押すと、モーターは最大速度および最大加速度で目標位置に到達します。モーターが最高速で動作するので十分注意してください。

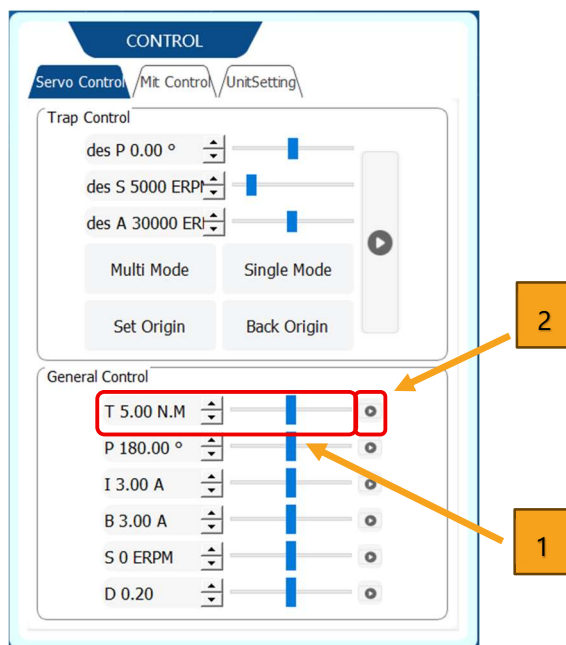


図 20 Position Loop モード

3.9.5. Current Loop(電流フィードバック)モード

「Servo Control」タブの「General Control」セクション画面で目標電流 I を入力して右のスタートボタンを押すと、モーターは目標電流で動作します。無負荷または低負荷時は回転してしまうので注意してください。

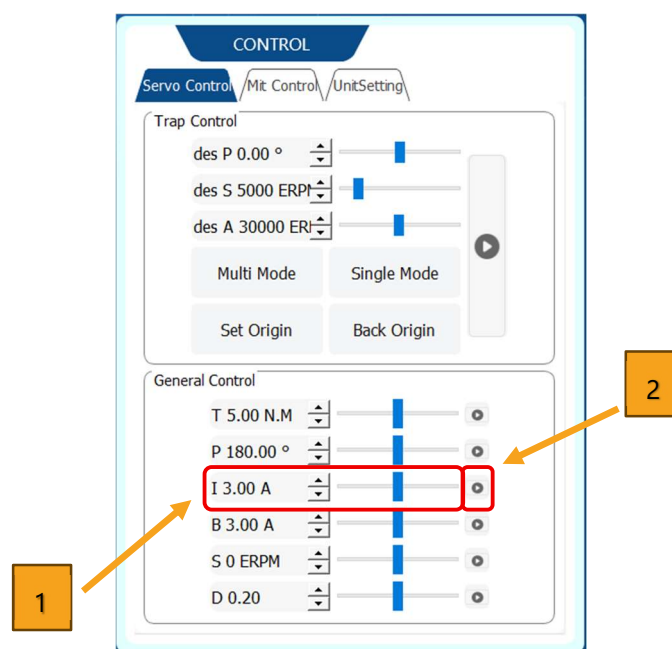


図 21 Current Loop モード

3.9.6. Velocity Loop(速度フィードバック)モード

「Servo Control」タブの「General Control」セクション画面で目標速度 S（±50000 ERPM）を入力して右のスタートボタンを押すと、モーターは目標速度で動作します（デフォルトでは最大加速度）。

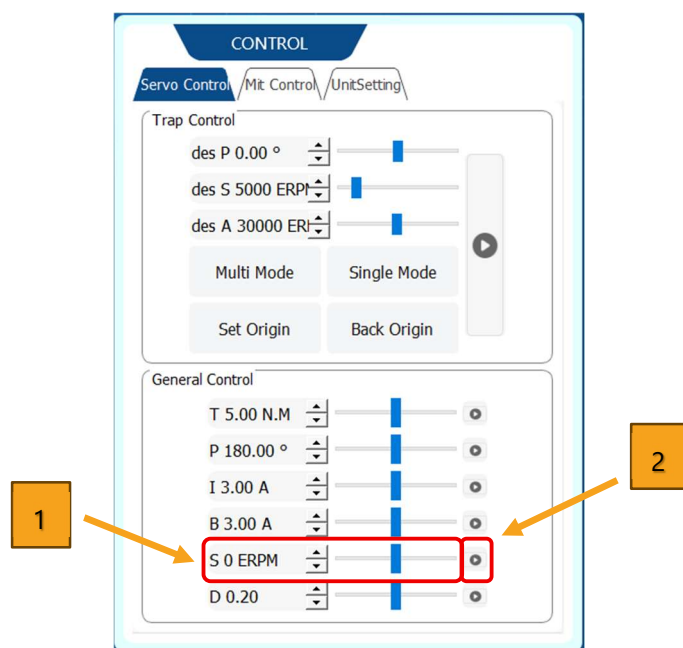


図 22 Velocity Loop モード

3.9.7. Braking Loop(ブレーキ)モード

「Servo Control」タブの「General Control」セクション画面で目標ブレーキ電流 B を入力して右のスタートボタンを押すと、モーターは目標電流でブレーキをかけます。

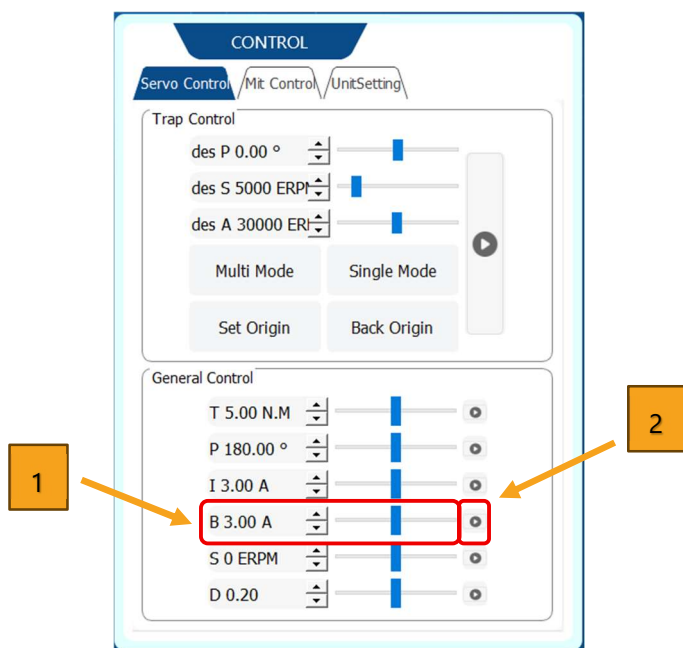


図 23 Braking Loop モード

3.9.8. Duty Cycle(デューティ比入力)モード

「Servo Control」タブの「General Control」セクション画面で目標デューティサイクル（デフォルトは 0.005～0.95）を入力して右のスタートボタンを押すと、モーターは目標デューティ比で動作 * します。

* 最大（最小）速度を ± 1.00 としたときの速度指令値

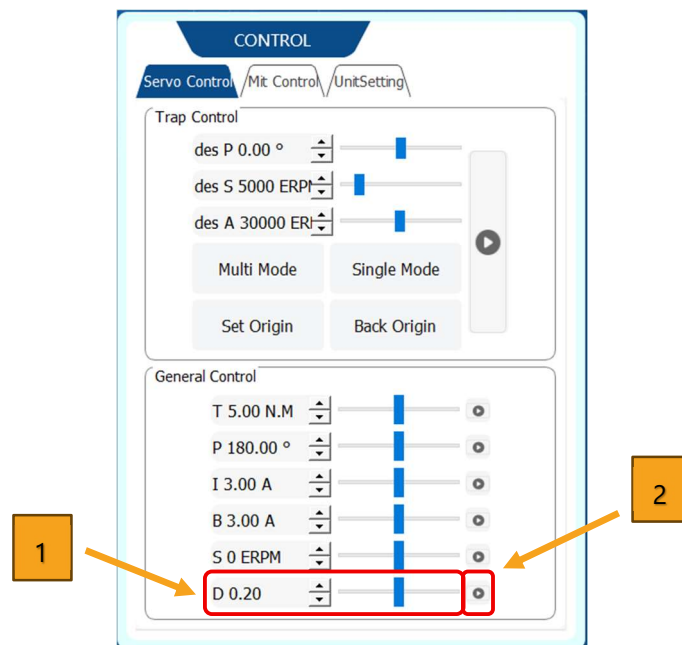


図 24 Duty Cycle モード

3.9.9. Force Control(力制御・MIT)モード

「MIT Control」タブの「Force Control」セクション画面で対応する「CAN ID」、目標位置、Kp、Kd を入力し、「Start」をクリックすると、モーターは目標位置へ移動します。モーターは指定された位置および Kp、Kd 値で動作し、指定位置へ向かって回転します。

Force Control Position(位置・速度制御)モード

Kp および Kd に 0 以外の値を入力します。目標速度 des S は数値を入力しても 0 として扱われます。一般的に Kp を大きくすると素早く・精度よく動作し、Kd を大きくするとゆっくり動作しますが、Kp と比較して Kd の値が小さすぎると振動して目標位置に到達しないことがあります。

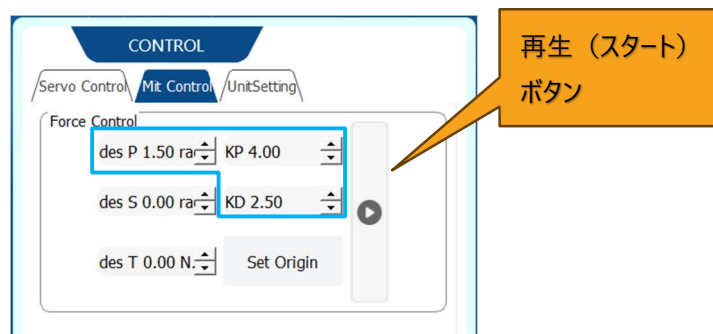


図 25 Force Control 位置・速度制御モード

Force Control Velocity(速度制御)モード

Kp に 0 を入力し、Kd に 0 以外の値を入力します。

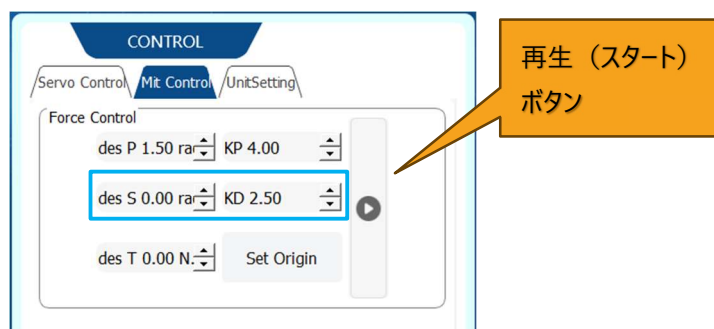


図 26 Force Control 速度制御モード

Force Control Torque(カフィードバック制御)モード

Kp、Kd に関係なく des T に 0 以外の値を入力すると、トルク T (Nm) が発生します。無負荷または低負荷時はモーターが回転し続けるので注意してください。

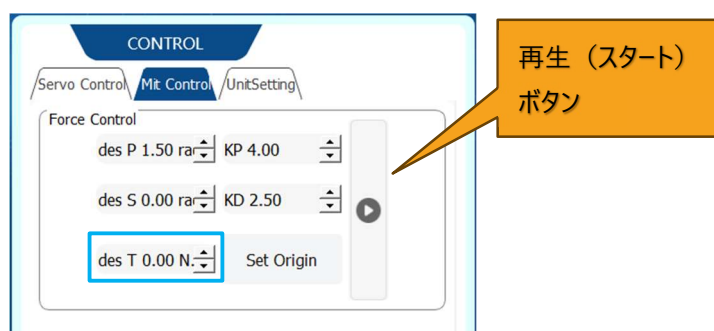


図 27 Force Control Torque モード

3.10. ファームウェアアップグレード

ファームウェアをアップグレードする際には電源や接続などが外れたりしないよう、十分に気をつけてください。

1. 「Choose Fw」ドロップダウンリストから対応するファームウェアを選択します。
2. 「Jump to IAP」ボタンをクリックします。
3. 「Upload」ボタンをクリックし、アップグレード進捗バーが 100%に達するまで待ちます。
4. 「Jump to App」ボタンをクリックし、5 秒待つと、モーターは動作モードに入り、ファームウェア更新が完了します。

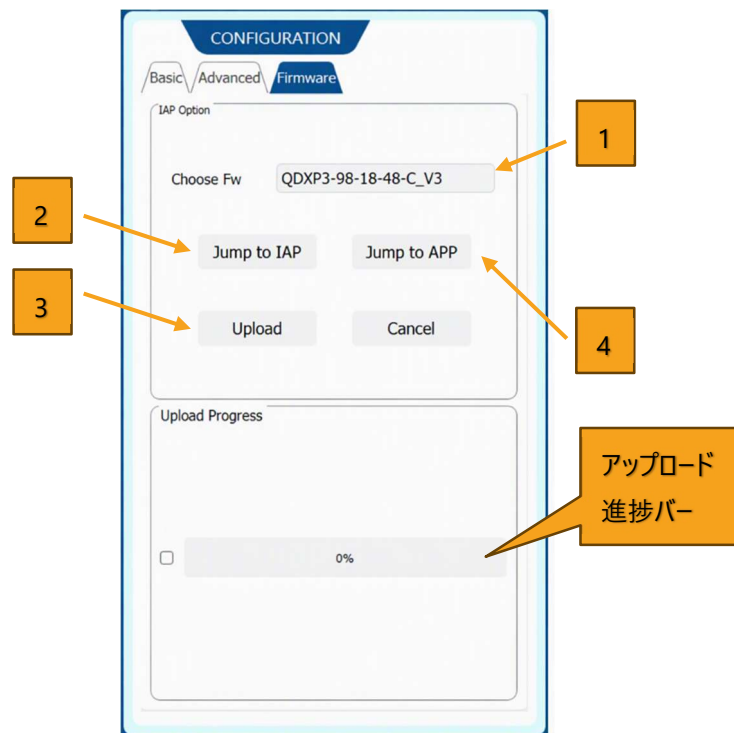


図 28 ファームウェアアップグレード

4. ドライバー基板通信プロトコル

4.1. サーボ制御モード

サーボモードには 6 種類の制御モードがあり、それぞれ特定の ID 番号に対応しています。

Duty Cycle Mode (デューティサイクルモード) :

指定されたデューティサイクル電圧をモーターに与えるモードで、PWM 駆動形式に類似しています。

Current Loop Mode (電流ループモード) :

電流値 i_q を指定してモーターに与えます。モーター出力トルク $= i_q \times K_T$ であるため、トルクループとして使用できます (K_T データは電流当たりのトルクを示す定数で、モーター個別の値となります。)。

Current Brake Mode (電流ブレーキモード) :

指定されたブレーキ電流をモーターに与え、現在位置を保持します (使用時はモーター温度に注意してください)。

Velocity Mode (速度モード) :

指定された動作速度をモーターに与えます (加速度はデフォルトで最大値です)。

Position Mode (位置モード) :

指定された位置をモーターに与えると、モーターは指定位置へ移動します (速度および加速度はデフォルトで最大値です)。

Position-Velocity Loop Mode (位置・速度ループモード) :

指定された位置、速度、加速度をモーターに与えます。

サーボモードデータ送信定義

Identifier (識別子) :

Control Mode ID (制御モード ID) + Driver ID (ドライバー ID = モーター ID 番号) 制御モード ID + ドライバー ID

Frame Type (フレーム形式) :

CAN フレームは 8 bytes の拡張フレーム形式を使用します。DATA DLC: 8byte 拡張フレーム形式は下記のようになります。

Can ID bits	[28]-[8]	[7]-[0]
フィールド名 (機能定義名)	制御モード (Control ModeID)	モーター ID (ドライバー基板 ID)

Control mode ID (制御モード) には {0, 1, 2, 3, 4, 5, 6, 15, 16} の設定値があり、9 種類の制御モードに対応します。

Duty Cycle Mode (デューティサイクルモード) : 0

Current Loop Mode (電流ループ・電流フィードバックループモード) : 1

Current Brake Mode (ブレーキモード) : 2

Velocity Mode (速度制御モード) : 3

Position Mode（位置制御モード）：4

Set Origin Mode（原点設定モード）：5

Position-Speed Loop Mode（位置・速度制御モード）：6

Motor disable mode（モーターOFF）：15

Feedback message configuration（メッセージ設定）：16

以下では、各種モードでモーターを制御する例を示します。以下は、各種インスタンス用のライブラリ呼び出し、関数、およびマクロ定義です。

本マニュアルにおける C 言語コード例について

- サンプルコードは Arduino 用です。一般的な関数はそのまま C 言語のソースとして使用できますが、デバイス間通信（CAN など）はお客様の環境に応じて書き換える必要があります。
- モーターの動作テストでは①M5Stack Basic、②M5Stack Commu モジュール、③GROVE ケーブル、④Grove スクリューターミナル（Grove 端子台）を使用しました。
- CAN トランシーバは MCP2515-1 / SO を使用しています。他の CAN トランシーバを使う場合はソースコードを書き換えてください。

```
typedef enum {
    CAN_PACKET_SET_DUTY = 0, // Duty Cycle Mode
    CAN_PACKET_SET_CURRENT, // Current Loop Mode
    CAN_PACKET_SET_CURRENT_BRAKE, // Current Brake Mode
    CAN_PACKET_SET_RPM, // Speed Mode
    CAN_PACKET_SET_POS, // Position Mode
    CAN_PACKET_SET_ORIGIN_HERE, // Set origin position mode (zero mode)
    CAN_PACKET_SET_POS_SPD, // Position-Velocity Loop Mode
    CAN_PACKET_SET_MIT = 8, // Force Control (MIT) Mode
    CAN_DISABLE_ = 15, //Motor disabled
    CAN_Frame_ = 16 //Feedback message configuration
} CAN_PACKET_ID;

/// @brief CAN メッセージを送信します
/// @param id モーターID 番号を指定します
/// @param data メッセージ
/// @param len CAN メッセージのデータ長（最大 8 バイト）
void comm_can_transmit_eid(uint32_t id, const uint8_t *data, uint8_t len) {
    uint8_t i = 0;

    if (len > 8) {
        len = 8;
    }

    CanTxMsg TxMessage;
    TxMessage.StdId = 0;
    TxMessage.IDE = CAN_ID_EXT;
    TxMessage.ExtId = id;
    TxMessage.RTR = CAN_RTR_DATA;
```

```

TxMessage.DLC = len;

for (i = 0; i < len; i++) {
    TxMessage.Data[i] = data[i];
}

CAN_Transmit (CHASSIS_CAN, &TxMessage); // CAN ポートからメッセージを送信します
}

/// @brief バッファの任意の位置へ 32bit int を代入します
/// @param buffer 元データ
/// @param number 変換元の int32 型パラメータ
/// @param index number を置き換える buffer のインデックス位置
void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index) {
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

/// @brief バッファの任意の位置へ 16bit int を代入します
/// @param buffer 元データ
/// @param number 変換元の int16 型パラメータ
/// @param index number を置き換える buffer のインデックス位置
void buffer_append_int16(uint8_t* buffer, int16_t number, int16_t *index) {
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

```

4.1.1. デューティサイクルモード(Duty Cycle Mode)

デューティサイクルモードのデータ送信定義は次の通りです。

データビット	Data[0]	Data[1]	Data[2]	Data[3]
範囲	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	デューティ比 25-32bit	デューティ比 17-24bit	デューティ比 9-16bit	デューティ比 1-8bit

デューティ比は int32 型で、-100000~100000 の指令値は-100~100 (%) を表します。

デューティサイクルモードコマンド作成関数例

```

/// @brief Duty Cycle Mode
/// @param controller_id モーター（ドライバー）ID 番号を指定します
/// @param duty デューティ比
void comm_can_set_duty(uint8_t controller_id, float duty) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(duty * 100000.0), &send_index);
}

```

```
comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_DUTY << 8), buffer,
send_index);
}
```

4.1.2. 電流ループモード

電流ループモードのデータ送信定義は次の通りです。電流値を 4 バイトの整数に変換します。

データビット	Data[0]	Data[1]	Data[2]	Data[3]
範囲	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	電流値 25-32bit	電流値 17-24bit	電流値 9-16bit	電流値 1-8bit

電流値は int32 型で、-60000~60000 の指令値は-60~60 A を表します。

電流ループモードコマンド作成関数例

```
/// @brief 電流フィードバック制御メッセージを作成する
/// @param controller_id モーターID
/// @param current 電流値
void comm_can_set_current(uint8_t controller_id, float current) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);
    comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_CURRENT << 8), buffer,
send_index);
}
```

4.1.3. 電流ブレーキモード

電流ブレーキモードのデータ送信定義は次の通りです。

DataBit	Data[0]	Data[1]	Data[2]	Data[3]
範囲	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	ブレーキ電流値 25-32bit	ブレーキ電流値 17-24bit	ブレーキ電流値 9-16bit	ブレーキ電流値 1-8bit

ブレーキ電流値は unsigned int32 型です。数値の範囲は 0~60000 で、実際の値は 0~60 A となります。

電流ブレーキモード送信サンプルルーチン

```
/// @brief 電流ブレーキモード
/// @param controller_id モーターID
/// @param current ブレーキ電流値
void comm_can_set_cb(uint8_t controller_id, float current) {
    int32_t send_index = 0;
    uint8_t buffer[4];
```

```

buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);
comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_CURRENT_BRAKE << 8),
    buffer, send_index);
}

```

4.1.4. 速度ループモード

速度ループモードのデータ送信定義は次の通りです。

DataBit	Data[0]	Data[1]	Data[2]	Data[3]
範囲	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	速度値 25-32bit	速度値 17-24bit	速度値 9-16bit	速度値 1-8bit

速度値は int32 型で、数値の範囲は-100000~100000 で、実際の値は-100000-100000ERPM となります。

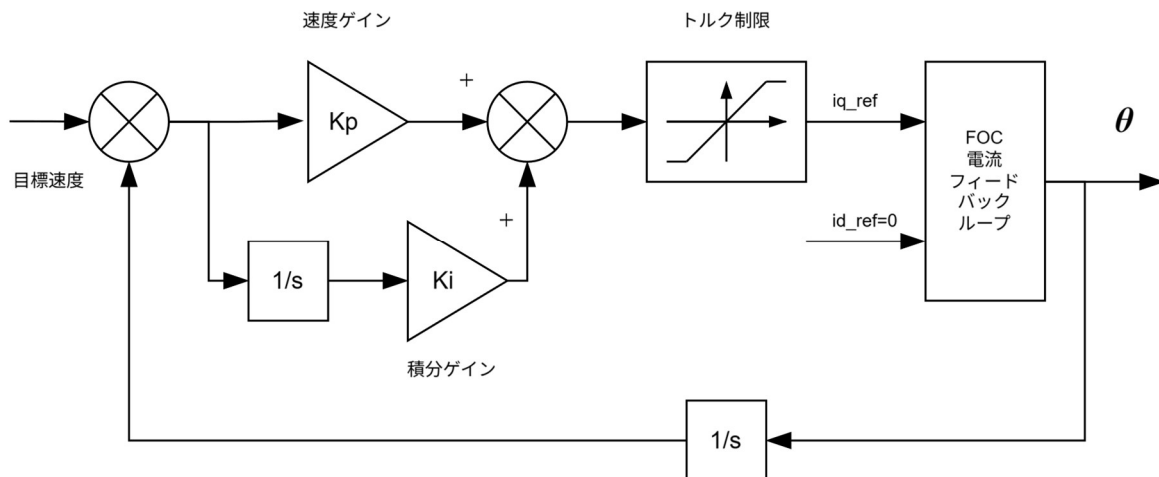


図 29 速度ループ制御ブロック図

速度ループ送信サンプルルーチン

```

/// @brief 速度設定
/// @param controller_id モーターID
/// @param rpm 速度(ERPM)
void comm_can_set_rpm(uint8_t controller_id, float rpm) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)rpm, &send_index);
    comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_RPM << 8), buffer,
        send_index);
}

```

4.1.5. 位置ループ(位置フィードバック)モード

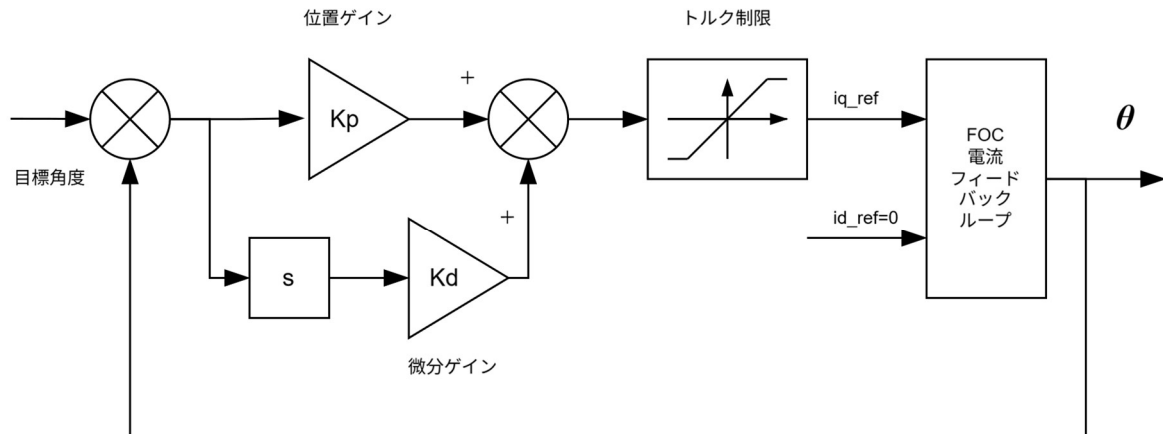


図 30 位置フィードバックループ制御ブロック図

位置ループモードのデータ送信定義

Data Bit	Data[0]	Data[1]	Data[2]	Data[3]
範囲	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	角度値 25-32bit	角度値 17-24bit	角度値 9-16bit	角度値 1-8bit

位置値は int32 型で、範囲-3600000000~3600000000 は-36000°~36000°の角度を表します。

位置ループ送信サンプルルーチン

```

/// @brief 位置制御モード
/// @param controller_id モーターID
/// @param pos 目標位置
void comm_can_set_pos(uint8_t controller_id, float pos) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_POS << 8), buffer,
    send_index);
}

```

4.1.6. 原点設定モード

設定コマンドは uint8_t 型で、0 は一時原点の設定（電源断で消去）、1 は永久原点の設定（パラメータは自動保存）を表します。

Data Bit	Data[0]
範囲	0~0x01
対応する変数	0 または 1

原点設定コマンドサンプルルーチン

```

/// @brief 原点設定モード
/// @param controller_id モーターID

```

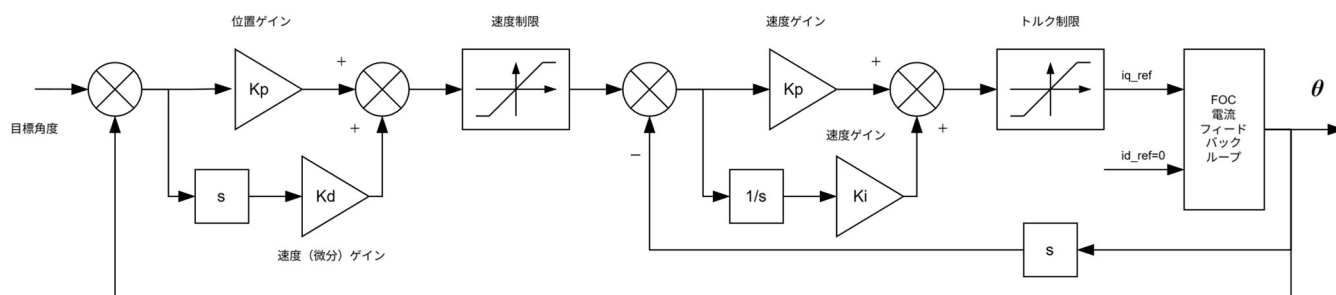
```

/// @param set_origin_mode
void comm_can_set_origin(uint8_t controller_id, uint8_t set_origin_mode) {
    int32_t send_index = 1;
    uint8_t buffer[1];
    buffer[0] = set_origin_mode;
    comm_can_transmit_eid(controller_id | ((uint32_t) CAN_PACKET_SET_ORIGIN_HERE << 8), buffer,
    send_index);
}

```

4.1.7. 位置・速度ループモード

位置・速度ループの簡略図



位置・速度ループモードのデータ送信コマンド定義

Data Bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
範囲	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	角度値 25-32bit	角度値 17-24bit	角度値 9-16bit	角度値 1-8bit	速度値 上位1Byte	速度値 下位1Byte	加加速度値 上位1Byte	加加速度値 下位1Byte

位置値は int32 型で、範囲-360000000~360000000 は-36000°~36000°の位置に対応します。

速度値は int16 型で、範囲-32768~32767 は-327680~327680 ERPM に対応します。

加加速度値は int16 型で、範囲 0~32767 は 0~327670 に対応し、1 単位は 10 ERPM/s²に相当します。

位置・速度制御コマンド関数例

```

/// @brief 位置・速度制御モード
/// @param controller_id モーターID
/// @param pos 目標位置
/// @param spd 目標速度
/// @param RPA 目標加減速度
void comm_can_set_pos_spd(uint8_t controller_id, float pos, int16_t spd, int16_t RPA ) {
    int32_t send_index = 0;
    int16_t send_index1 = 4;
    uint8_t buffer[8];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    buffer_append_int16(buffer, spd/10.0, &send_index1);
    buffer_append_int16(buffer, RPA/10.0, &send_index1);
}

```

```
comm_can_transmit_eid(controller_id | ((uint32_t)CAN_PACKET_SET_POS_SPD << 8), buffer,
send_index1);
}
```

4.1.8. モーター無効化(モーターOFF)モード

送信するデータ内容は不要です。対応する機能フレームを送信するだけでモーターを無効化できます。

モーターは ID 0x29 のフレームの DATA[7]を 0x77 に設定して 1 回返し、無効化が成功したことを示します。

```
/// @brief モーターOFF
/// @param controller_id モーターID
void comm_can_disable(uint8_t controller_id) {
    int16_t send_index1 = 0;
    uint8_t buffer[8];
    comm_can_transmit_eid(controller_id | ((uint32_t)CAN_DISABLE_ << 8), buffer, send_index1);
}
```

4.1.9. フィードバックメッセージ設定

Data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
範囲	0	0	0	0	0	0	0~0xff	0~0xff
対応する 変数	Keep	Keep	Keep	Keep	Keep	Keep	変数の上 位1バイト	変数の下 位1バイト

△注意：このコマンドは Flash メモリへ書き込みます。頻繁に実行しないでください。

パラメータの対応ビットが 1 の場合、機能が有効になります。0 にすると無効となります。パラメータは電源を切っても保持されます。

パラメータの Bit [15]：

0x2A (4 バイト位置) フィードバックフレームを追加します。

パラメータの Bit [14]：

単回転モード、位置範囲 0~360°。その他のビット：予約。

たとえば、パラメータ値 0x8000 は、4 バイト位置フィードバックが追加されることを示します。

```
/// @brief フレーム設定
/// @param controller_id モーターID
/// @param frame
void comm_can_set_frame(uint8_t controller_id, uint16_t frame) {
    int32_t send_index = 0;
    int16_t send_index1 = 6;
    uint8_t buffer[8];
    buffer_append_uint16(buffer, frame, & send_index1);
    comm_can_transmit_eid(controller_id | ((uint32_t)CAN_Frame_ << 8), buffer, send_index1);
}
```

4.2. 力制御モード(MIT モード)通信プロトコル

力制御モードには 3 種類の制御モードがあり、すべて同一のモーターID を使用します。具体的な制御モードは送信データによって決まります。各モードの MISUMI QDXP Motor Tool での詳細操作については、3.9.9 節を参照してください。

Position Loop Mode (位置フィードバックループモード) :

指定された位置および Kp、Kd 値でモーターを指定位置へ回転します。

Velocity Loop Mode (速度フィードバックループモード) :

指定された速度および Kd 値でモーターを指定速度で回転します。Kp を 0 にします。

Current Loop Mode (電流フィードバックループモード) :

指定されたトルク値でモーターを動作させ、モーターは指定トルク値で回転します。

力制御モードは CAN プロトコルを使用し、以下に示す拡張フレーム形式を採用します。

Can ID bits	[28]-[8]	[7]-[0]
Field name	Control mode ID	Drive ID (モーターID)

The Control Mode ID for the **Force Control Mode** has a characteristic value of 8

力制御モードの制御モード ID の設定値 (ModeID) は 8 です。

力制御モード ドライバー基板データ送信定義

識別子 :

制御モード ID + ドライバーID

フレームタイプ :

拡張フレーム

フレーム形式 :

データ 8 バイト

Data Field	DATA[0]	DATA[1]		DATA[2]	DATA[3]
Data Bits	7-0	7-4	3-0	7-0	7-0
データ内容	上位 8bit KP	下位 4bit KP	上位 4bits KD	下位 8bits KD	上位 8bits 角度(rad)
Data Field	DATA[4]	DATA[5]	DATA[6]		DATA[7]
Data Bits	7-0	7-0	7-4	3-0	7-0
データ内容	下位 8bit 角度(rad)	上位 8bit 角度(rad)	上位 4bits 速度(rad/s)	下位 4bits 電流値(A)	上位 8bits 電流値(A)

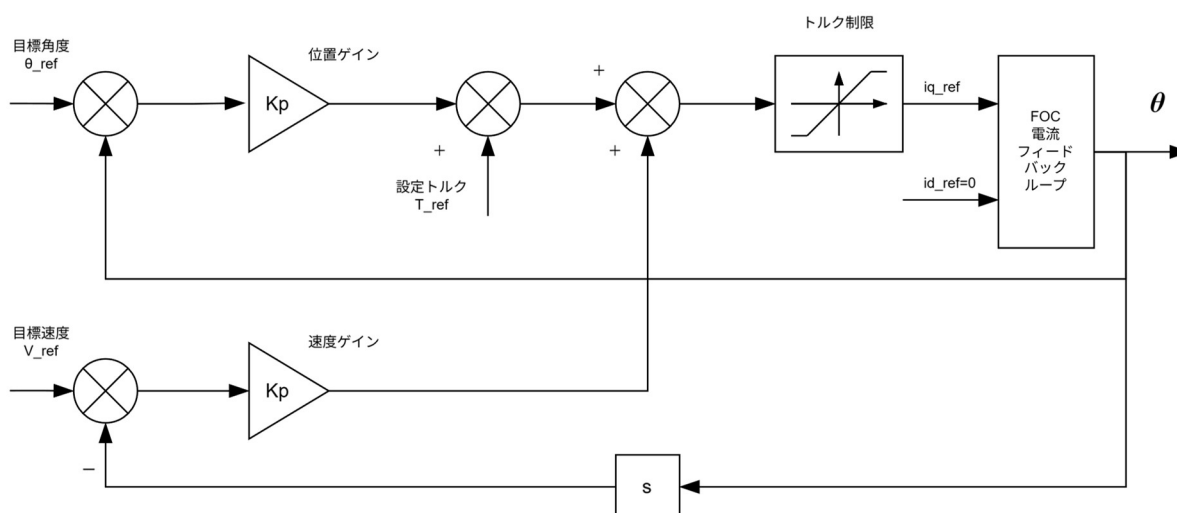


図 31 力制御（MIT）モード制御ブロック図

MIT モードでは単精度浮動小数点を使用するため、送信データ（バイト列）と実データ（単精度浮動小数）の変換には、各変数の最小値と最大値を使用します。表はモーターの各変数の最大値と最小値を表にまとめたものです。表にある各変数（P_MIN など）はプログラムで使用します

表 7 モーターパラメータ範囲

モーター 型番	KV	モーター角度 P_MIN/P_MAX (rad)	トルク定数 (Nm/A)	モーター速度 V_MIN/V_MAX (rad/s)	トルク T_MIN/T_MAX (Nm)	位置 (比例) ゲイン Kp	速度 (微分) ゲイン Kd
QDXP1-53-2.5-24-C	75	-12.5f～ 12.5f	0.127	-0.60f～ 0.60f	-34.0f～ 34.0f	0～ 500.0f	0～ 5.0f
QDXP3-98-18-48-C	60	-12.5f～ 12.5f	1.3137	-28.0f～ 28.0f	-54.0f～ 54.0f	0～ 500.0f	0～ 5.0f
QDXP2-79-3-W-C	80	-12.5f～ 12.5f	0.5994	-60.0f～ 60.0f	-12.0f～ 12.0f	0～ 500.0f	0～ 5.0f
QDXP2-79-24-48-C	80	-12.5f～ 12.5f	3.4616	-10.0f～ 10.0f	-80.0f～ 80.0f	0～ 500.0f	0～ 5.0f
QDXP2-89-8.5-48-C	60	-12.5f～ 12.5f	1.0621	-30.0f～ 30.0f	-32.0f～ 32.0f	0～ 500.0f	0～ 5.0f
QDXP2-98-9-48-C	100	-12.5f～ 12.5f	0.5701	-65.0f～ 65.0f	-18.0f～ 18.0f	0～ 500.0f	0～ 5.0f
QDXP4-90-26-48-C	41	-12.5f～ 12.5f	2.8334	-13.0f～ 13.0f	-110.0f～ 110.0f	0～ 500.0f	0～ 5.0f
QDXP4-90-74-48-C	41	-12.5f～ 12.5f	8.8123	-5.0f～ 5.0f	-280.0f～ 280.0f	0～ 500.0f	0～ 5.0f

トルクと電流の関係は次のようになります。

$$T = K_t \times I_q$$

T :

出力軸トルク (Nm)

K_t :

トルク定数 (Nm/A)

I_q :

電流 (A)

MIT コマンドを使うサンプルコードです。

使用するモーターに合わせて P_MIN、P_MAX、V_MIN、V_MAX、T_MIN、T_MAX、Kp_MIN、Kp_MAX、Kd_MIN、Kd_MAX を書き換えてください。値は表 7 モーターパラメータ範囲を参照してください。

各制御に必要な変数を `void pack_cmd` 関数へ送ると、MIT コマンドのメッセージが生成できますので、`uint8_t SERVO_Can_Send_Msg` 関数でモーターへ MIT コマンド (`uint8_t* msg`) を送信してください。CAN 通信の処理はお使いのデバイスに合わせて書き換えてください。

```
uint8_t SERVO_Can_Send_Msg(uint32_t ExtId, uint8_t* msg, u8 len)
{
    uint8_t mbox;
    uint16_t i=0;
    // TxMessage.StdId=stdId; // standard identifier
    TxMessage.ExtId=ExtId; // Setting the Extended Marker
    TxMessage.IDE=CAN_Id_Extended; // standard frame
    TxMessage.RTR=CAN_RTR_Data; // data frame
    TxMessage.DLC=len; // Length of data to be sent

    for(i=0;i<len;i++) {
        TxMessage.Data[i]=msg[i];
    }
    mbox = CAN_Transmit(CANx, &TxMessage);
    i=0;

    while((CAN_TransmitStatus(CAN1, mbox)==CAN_TxStatus_Failed)&&(i<0xFFFF)) {
        i++;
    }
    // Waiting for the end of sending
    if(i>=0xFFFF) {
        return 1;
    }
    return 0;
}

float fmaxf(float x, float y){
    /// Returns maximum of x, y ///
    return (((x)>(y))?(x):(y));
}
```

```

}

float fminf(float x, float y){
    /// Returns minimum of x, y ///
    return (((x)<(y))?(x):(y));
}

float fmaxf3(float x, float y, float z){
    /// Returns maximum of x, y, z ///
    return (x > y ? (x > z ? x : z) : (y > z ? y : z));
}

float fminf3(float x, float y, float z){
    /// Returns minimum of x, y, z ///
    return (x < y ? (x < z ? x : z) : (y < z ? y : z));
}

float P_MIN =-12.56f; // 以下の変数はモーターごとに変更してください
float P_MAX =12.56f;
float V_MIN =-28.0f;
float V_MAX =28.0f;
float T_MIN =-54.0f;
float T_MAX =54.0f;
float Kp_MIN =0;
float Kp_MAX =500.0f;
float Kd_MIN =0;
float Kd_MAX =5.0f;
float Test_Pos=0.0f;

int p_int ;
int v_int ;
int kp_int ;
int kd_int ;
int t_int ;

/// @brief MIT モード CAN メッセージ作成関数
/// @param controller_id モーターID
/// @param p_des 目標位置(rad)
/// @param v_des 目標速度(rad/s)
/// @param kp 比例ゲイン
/// @param kd 微分ゲイン
/// @param t_ff フィードフォワードトルク(Nm)
void pack_cmd(uint8_t controller_id, float p_des, float v_des, float kp, float kd, float t_ff) {
    uint8_t buffer[8];
    p_des = fminf(fmaxf(P_MIN, p_des), P_MAX);
    v_des = fminf(fmaxf(V_MIN, v_des), V_MAX);
    kp = fminf(fmaxf(Kp_MIN, kp), Kp_MAX);
    kd = fminf(fmaxf(Kd_MIN, kd), Kd_MAX);
    t_ff = fminf(fmaxf(T_MIN, t_ff), T_MAX);
}

```



```

/// convert floats to unsigned ints ///
p_int = float_to_uint(p_des, P_MIN, P_MAX, 16);
v_int = float_to_uint(v_des, V_MIN, V_MAX, 12);
kp_int = float_to_uint(kp, Kp_MIN, Kp_MAX, 12);
kd_int = float_to_uint(kd, Kd_MIN, Kd_MAX, 12);
t_int = float_to_uint(t_ff, T_MIN, T_MAX, 12);

/// pack ints into the can buffer ///
buffer[0] = kp_int>>4; //KP high 8 bits
buffer[1] = ((kp_int&0xF)<<4)|( kd_int>>8); //KP Low 4 bits, Kd High 4 bits
buffer[2] = kd_int&0xFF; //Kd low 8 bits
buffer[3] = p_int>>8; //position high 8 bits
buffer[4] = p_int&0xFF; //position low 8 bits
buffer[5] = v_int>>4; //speed high 8 bits
buffer[6] = ((v_int&0xF)<<4)|(t_int>>8); //speed low 4 bits torque high 4 bits
buffer[7] = t_int&0xFF; //torque low 8 bits
SERVO_Can_Send_Msg(controller_id |((uint32_t) CAN_PACKET_SET_MIT << 8), buffer, 8);
}

```

すべての数値は下記のプログラムによって整数値に変換してからモーターに送信してください

```

int float_to_uint(float x, float x_min, float x_max, unsigned int bits){
    /// Converts a float to an unsigned int, given range and number of bits ///
    float span = x_max - x_min;
    if(x < x_min) {
        x = x_min;
    }
    else if(x > x_max) {
        x = x_max;
    }
    return (int) ((x- x_min)*((float)((1<<bits)/span)));
}

```

4.3. モーターCAN メッセージフォーマット

4.3.1. CAN 送信(ホスト→モーター)メッセージプロトコル

サーボモードではドライバーへの CAN 送信メッセージは次のようなフォーマットになります

Identifier (CAN メッセージ識別子) :

Function ID + Driver ID (下記の Function ID とモーターID を足し合わせたもの)

Frame type (CAN フレームタイプ) :

Extended frame (拡張フレーム)

Frame format (フレームフォーマット) :

データ

DATA DLC (データ長さ) :

8 bytes

サーボモードでは下記のような拡張 CAN フレーム (データ内容のこと) を使用します。

Can ID bits	[28]-[8]	[7]-[0]
フィールド名	Function ID (機能ID)	Drive ID (モーターID)

Function ID は次の 2 種類があります。

0x2C :

0x2C はモーターがサーボモードになる際の開始フレームです。

モーターからの応答データは 0xFA 0xFB 0xFC 0xFD に固定されています。

0x29 :

0x29 はサーボモードにおけるモーターの現在の状態 (データバイトを含む) を受け取るリアルタイムのフィードバックを表します。

モーターのリアルタイムステータスは一定周期ごとにドライバーからホストへアップロードされます (CAN モードが Periodic Feedback のとき)。その周期は 1 から 2000Hz (0.5ms) の間で設定可能です。

Data Bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
範囲	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
対応する変数	角度上位 1byte	角度下位 1byte	速度上位 1byte	速度下位 1byte	電流上位 1byte	電流下位 1byte	モーター 温度	エラー コード

角度 (Position) フレーム : Data[0]、Data[1]

int16 型整数です。-32000～32000 の範囲で指定可能で、-3200～3200°に対応します。

速度 (Speed) フレーム： Data[2]、Data[3]

int16 型整数です。-32000～32000 の範囲で数値を指定可能で、-320000～320000ERPM に対応します。

電流値 (Current) フレーム： Data[4]、Data[5]

int16 型整数で-6000 から 6000 の範囲で数値を指定可能です。-60A～60A に対応します。

モーター基板温度 (Temperature) フレーム： Data[6]

int8 型整数で、-20～127 の値になります。-20°C～127°Cに対応します。

エラーコード (Error Code) フレーム： Data[7]

unsigned int8 型の整数です。各数値は下記のようになります。

- 0： エラーなし
- 1： モーター温度が高すぎる
- 2： 過電流
- 3： 過電圧
- 4： 低電圧
- 5： エンコーダエラー
- 6： モータードライバー基板の MOSFET 温度が高すぎる
- 7： モーターロック（外力などでモーターが回らなくなった）

下記のコードはモーターからのメッセージを読み取る関数例です。

```
void motor_receive (float* motor_pos,float* motor_spd,float* motor_cur,int8_t* motor_temp, int8_t*  
motor_error, int8_t* rx_message)  
{  
    int16_t pos_int = ((rx_message)->Data[0] << 8) | ((rx_message)->Data[1]);  
    int16_t spd_int = ((rx_message)->Data[2] << 8) | ((rx_message)->Data[3]);  
    int16_t cur_int = ((rx_message)->Data[4] << 8) | ((rx_message)->Data[5]);  
    *motor_pos = (float)( pos_int * 0.1f); //Motor position  
    *motor_spd = (float)( spd_int * 10.0f); //Motor speed  
    *motor_cur = (float) ( cur_int * 0.01f); //Motor current  
    *motor_temp= (rx_message)->Data[6]; //Motor temperature  
    *motor_error = (rx_message)->Data[7]; //Motor error code  
}
```

0x2A(オプション)

モータードライバーから来るメッセージの設定命令で 0x2A に設定するとモーターポジション情報が int32 型 4 バイトに拡張されます。範囲は-2,147,483,648～2,147,483,647 で-21,474,836.48° ～21,474,836.47° に対応します。

4.3.2. シリアルポートメッセージプロトコル

シリアルポートの送受信メッセージプロトコルは以下のとおりです。

フレーム ヘッダ	データ長	データフレーム	データ	チェックサム 2byte		フレーム フッター
0xAA	フレームヘッダ・フッタ・チェックサムを除く	データ定義を参照	...	上位 1byte	下位 1byte	0xBB

チェックサムの計算については、4.3.5 節を参照してください。

以下はデータフレームの定義をプログラムコードで表したものです。

```
typedef enum {
    COMM_FW_VERSION = 65,
    COMM_JUMP_TO_BOOTLOADER = 66,
    COMM_ERASE_NEW_APP = 67,
    COMM_WRITE_NEW_APP_DATA = 68,
    COMM_GET_VALUES = 69, // モーターパラメータの取得

    COMM_SET_DUTY = 70, // デューティサイクルモード
    COMM_SET_CURRENT = 71, // 電流フィードバックモード
    COMM_SET_CURRENT_BRAKE = 72, // 電流ブレーキモード
    COMM_SET_RPM = 73, // 速度フィードバックモード
    COMM_SET_POS = 74, // 位置制御モード
    COMM_SET_HANDBRAKE = 75, // 電流ハンドブレーキモード
    COMM_SET_DETECT = 76, // 現在位置の取得
    COMM_ROTOR_POSITION = 87, // 現在のローター角度の取得
    COMM_GET_VALUES_SETUP = 16, // The motor requires instructions based on one or more
parameters

    COMM_SET_POS_SPD = 60, // 位置・速度制御モード
    COMM_SET_POS_MULTIPLE = 61, // モーターを多重回転モードにする ±100
    COMM_SET_POS_SINGLE = 62, // モーターを単回転モードにする
    COMM_SET_POS_UNLIMITED = 63, // パラメータの保存
    COMM_SET_POS_ORIGIN = 64, // 原点設定
} COMM_PACKET_ID;
```

4.3.3. パラメータの取得

以下の二桁の数値を空白で区切っている数値表現（例：01 18）は 0x を省略した 16 進数です。

モーターパラメータの取得例

送信コマンド : COMM GET VALUES (0x45=69)

AA 01 45 18 61 BB

モーターからの返事：

AA 55 45 01 84 01 18 00 01

BE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 02 00 43 07 5A 9D
68 00 00 00 00 00 00 FF FF FF FF FF FF FD 00 00 00 0A 43 0A AE 30 43 9F 01 D3 DC FD BB

説明：

モーターパラメータを取得します（1 度の返事で全データがモーターから戻ります）

受信データ例と内容：

AA（フレームヘッダー） + 55（データ長） + 45（データフレーム） + MOS 温度（2byte） + モーター温度（2byte） + 出力電流（4byte） + 入力電流（4byte） + Id 電流（4byte） + Iq 電流（4byte） + モータースロットル値（2byte） + モーター速度（4byte） + 入力電圧（2byte） + 予約値（24byte） + モーター状態コード（1byte） + モーター外側ループ位置値（4byte） + モーター制御 ID 番号（1byte） + 温度予約値（6byte） + Vd 電圧値（4byte） + Vq 電圧値（4byte） + 電流制御モード（4byte） + エンコーダ角度（4byte） + 外部エンコーダ角度（4byte） + CRC + BB（フレーム終端）

モーターから送信される一部パラメータの変換式は以下のとおりです。

```
MOS temperature = (float)buffer_get_int16(data, &ind) / 10.0f;
Motor temperature = (float)buffer_get_int16(data, &ind) / 10.0f;
Output current = (float)buffer_get_int32(data, &ind) / 100.0f;
Input current = (float)buffer_get_int32(data, &ind) / 100.0f;
Id current = (float)buffer_get_int32(data, &ind) / 100.0f;
Iq current = (float)buffer_get_int32(data, &ind) / 100.0f;
Duty cycle value = (float)buffer_get_int16(data, &ind) / 1000.0f;
Motor speed = (float)buffer_get_int32(data, &ind);
Input voltage = (float)buffer_get_int16(data, &ind) / 10.0f;
Motor outer loop position = (float)buffer_get_float (data, &ind);
Motor ID number = data;
Motor Vd voltage = (float)buffer_get_int32 (data, &ind) / 1000.0f;
Motor Vq voltage = (float)buffer_get_int32(data, &ind) / 1000.0f;
Encoder angle = (float)buffer_get_float (data, &ind);
Outer encoder angle = (float)buffer_get_float (data, &ind);
```

モーター位置の取得

送信コマンド：COMM_SET_DETECT（0x4C=76）

AA 02 4C 04 08 25 BB

モーターからの返事：

AA 05 57 00 1A B6 64 6E CD BB

補足説明：

モーターは受信後 10ms ごとに現在位置を送信します（デフォルト設定時）。

パラメータ解析：

```
Pos = (float)buffer_get_float (data, &ind);
```

モーターの単一または複数パラメータの取得

送信コマンド：

AA 05 13 00 00 00 01 FA A9 BB //Get MOS temperature

モーターからの返事：

AA 07 13 00 00 00 01 01 21 DF BB // Feedback motor temperature

補足説明：

単一または複数のモーターパラメータを取得します。パラメータはコマンドデータ領域内の 4 バイト（32 ビット）によって決まります。対応するビットが 1 の場合、モーターはそのビットに対応するモーターパラメータを返し、0 の場合、そのフィールドは除外されます（モーターパラメータの取得順序は最初のビットから始まります）。

各ビットに対応するモーターパラメータを以下の表に示します。

Data byte(s)	31-18	17	16	15	14-9	8	7	6
内容	Reserved	モーターID 1byte	モーター 角度 4byte	エラー フラグ 1byte	Reserved	入力電圧 2byte	モーター速 度 (RPM) 4byte	デューティ サイクル 2byte
Data byte(s)	5	4	3	2	1	0		
内容	Iq電流 4byte	Id電流 4byte	入力電流 4byte	出力電流 4byte	モーター 温度 2byte	MOS 温度 2byte		

モーターから送信される一部パラメータの変換式は以下のとおりです。

MOS temperature = (float)buffer_get_int16(data, &ind) / 10.0

MOS 温度 = (float)buffer_get_int16(data, &ind) / 10.0

Motor temperature = (float)buffer_get_int16(data, &ind) / 10.0

Output current = (float)buffer_get_int32(data, &ind) / 100.0

Input current = (float)buffer_get_int32(data, &ind) / 100.0

Motor throttle value = (float)buffer_get_int16(data, &ind) / 1000.0

Motor speed = (float)buffer_get_int32(data, &ind)

Input voltage = (float)buffer_get_int16(data, &ind) / 10.0

Motor position = (float)buffer_get_int32(data, &ind) / 1000000.0

Motor ID number = data

Motor error status code

モーターエラー状態コード

```
typedef enum {
    FAULT_CODE_NONE = 0,
    FAULT_CODE_OVER_VOLTAGE, // 入力電圧が過電圧
```

```

FAULT_CODE_UNDER_VOLTAGE, // 入力電圧が低電圧
FAULT_CODE_DRV, // ドライバーエラー
FAULT_CODE_ABS_OVER_CURRENT, // モーター過電流
FAULT_CODE_OVER_TEMP_FET, // MOS FET の温度が高すぎる
FAULT_CODE_OVER_TEMP_MOTOR, // Motor の温度が高すぎる
FAULT_CODE_GATE_DRIVER_OVER_VOLTAGE, // ドライバー基板過電圧
FAULT_CODE_GATE_DRIVER_UNDER_VOLTAGE, // ドライバー基板低電圧
FAULT_CODE_MCU_UNDER_VOLTAGE, // MCU の電圧が低い
FAULT_CODE_BOOTING_FROM_WATCHDOG_RESET, // ウォッチドッグリセットからの起動
FAULT_CODE_ENCODER_SPI, // SPI エンコーダエラー
FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE, // エンコーダ制限値より低い
FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE, // エンコーダ制限値より大きい
FAULT_CODE_FLASH_CORRUPTION, // FLASH メモリー不正
FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_1, // 電流サンプルチャンネル 1 不正・失敗
FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_2, // 電流サンプルチャンネル 2 不正・失敗
FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_3, // 電流サンプルチャンネル 3 不正・失敗
FAULT_CODE_UNBALANCED_CURRENTS, // 電流が不均衡
} mc_fault_code;

```

4.3.4. 制御コマンド 1

デューティサイクル送信モード

送信コマンド:

```

AA 05 46 00 00 4E 20 D6 4C BB // 0.20 duty cycle
AA 05 46 FF FF B1 E0 88 3F BB // -0.20 duty cycle

```

受信パラメータ解析:

```

Duty = (float)buffer_get_int32(data, &ind) / 100000.0;
// The value is to receive 4 bytes of data/100000.0

```

電流ループ送信モード

送信コマンド:

```

AA 05 47 00 00 13 88 30 1C BB // 5A IQ current
AA 05 47 FF FF EC 78 58 3C BB // -5A IQ current

```

受信パラメータ解析:

```

Current = (float)buffer_get_int32(data, &ind) / 1000.0
// The value is to receive 4 bytes of data/1000.0

```

ブレーキループ電流送信モード

送信コマンド:

```

AA 05 48 00 00 13 88 55 E5 BB // 5A brake current
AA 05 48 FF FF EC 78 3D C5 BB // -5A brake current

```

受信パラメータ解析:

```

I_Brake = (float)buffer_get_int32(data, &ind) / 1000.0;

```

```
//The value is to receive 4 bytes of data/1000.0
```

速度ループ送信モード

送信コマンド：

```
AA 05 49 00 00 03 E8 90 61 BB // 1000 ERPM
```

```
AA 05 49 FF FF FC 18 F8 41 BB // - 1000 ERPM
```

受信パラメータ解析：

```
Speed=(float)buffer_get_int32(data, &ind)
//The value is to receive 4 bytes of data
```

位置ループ送信モード

送信コマンド：

```
AA 05 4A 0A BA 95 00 E1 4D BB //motor
```

rotates to 180°

```
AA 05 4A 05 5D 4A 80 84 93 BB //motor
```

rotates to 90°

パラメータ解析：

```
Pos=(float)buffer_get_int32(data, &ind) / 1000000.0
//The value is to receive 4 bytes of data/1000000.0
```

ハンドブレーキ電流送信モード

送信コマンド：

```
AA 05 48 00 00 13 88 55 E5 BB //5A HB Current
```

```
AA 05 48 FF FF EC 78 3D C5 BB //-5A HB Current
```

パラメータ解析：

```
HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
```

位置・速度ループ送信モード

送信コマンド：

```
AA 0D 3C 00 02 BF 20 00 00 13 88 00 00 75 30 18 1C BB
```

パラメータ解析：

Position (180°) + velocity (5000ERPM) + acceleration (3000ERPM/s)

```
Pos=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
Speed=(float)buffer_get_int32(data, &ind)
Acc_Speed=(float)buffer_get_int32(data, &ind)
//The value is to receive 4 bytes of data
```

MIT 送信モード

送信コマンド:

```
AA 15 60 00 00 00 00 00 00 17 70 00 00 00 00 00 00 00 00 07 D0 93 DA BB
// Kd set 2, speed set to 6rad/s
AA 15 60 00 00 00 00 FF FF E8 90 00 00 00 00 00 00 00 00 07 D0 87 5C BB
// Kd set 2, speed set to -6rad/s
```

パラメータ解析:

```
Position + Velocity + Torque + KP + KD
Pos=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
Speed=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
Current=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
KP=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
KD=(float)buffer_get_int32(data, &ind) / 1000.0
//The value is to receive 4 bytes of data/1000.0
```

データ変換関数:

```
int16_t buffer_get_int16(uint8_t* buffer, int32_t *index)
{
    int16_t res = ((uint16_t)buffer[*index])<<8 | ((uint16_t)buffer[*index+1]);
    return res;
}

uint16_t buffer_get_uint16(uint8_t* buffer, int32_t *index)
{
    uint16_t res = ((uint16_t)buffer[*index])<<8 | ((uint16_t)buffer[*index+1]);
    return res;
}

int32_t buffer_get_int32(uint8_t* buffer, int32_t *index)
{
    int32_t res = ((uint32_t)buffer[*index])<<24 | ((uint32_t)buffer[*index+1])<<16
        | ((uint32_t)buffer[*index+2])<<8 | ((uint32_t)buffer[*index+3]);
    return res;
}
```



```
uint32_t buffer_get_uint32(uint8_t* buffer, int32_t *index)
{
    uint32_t res = ((uint32_t)buffer[*index])<<24 | ((uint32_t)buffer[*index+1])<<16
        | ((uint32_t)buffer[*index+2])<<8 | ((uint32_t)buffer[*index+3]);
    return res;
}

float buffer_get_float(uint8_t* buffer, int32_t *index)
{
    float *res;
    uint32_t data;

    data = buffer_get_uint32(buffer,index); res = (float*)&data;
    return *res;
}

//int16 Data field layout
void buffer_append_int16 (uint8_t* buffer, int16_t number, int32_t *index)
{
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//uint16 Data field layout
void buffer_append_uint16 (uint8_t* buffer, uint16_t number, int32_t *index)
{
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//int32 Data field layout
void buffer_append_int32 (uint8_t* buffer, int32_t number, int32_t *index)
{
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//uint32 Data field layout
void buffer_append_uint32(uint8_t* buffer, uint32_t number, int32_t *index)
{
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}
```

4.3.5. シリアルポートキャリブレーション(serial port calibration)

```
const unsigned short crc16_tab[] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcdbc, 0xfbff, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

/// @brief CRC16 の計算
/// @param buf データ
/// @param len データ長
/// @return CRC チェックサム
unsigned short crc16(unsigned char *buf, unsigned int len)
{
    unsigned int i;
    unsigned short cksum = 0;
    for (i = 0; i < len; i++)
    {
        cksum = crc16_tab[(((cksum >> 8) ^ *buf++) & 0xFF)] ^ (cksum << 8);
    }
}
```

```

    }
    return cksum;
}

//Packet collation for sending
void packet_send_packet (unsigned char *data, unsigned int len, int handler_num)
{
    int b_ind = 0;
    unsigned short crc;
    if (len > PACKET_MAX_PL_LEN) {
        return;
    }
    if (len <= 256) {
        handler_states[handler_num].tx_buffer[b_ind++] = 0xAA;
        handler_states[handler_num].tx_buffer[b_ind++] = len;
    }
    else
    {
        handler_states[handler_num].tx_buffer[b_ind++] = 0xAB;
        handler_states[handler_num].tx_buffer[b_ind++] = len >> 8;
        handler_states[handler_num].tx_buffer[b_ind++] = len & 0xFF;
    }

    memcpy(handler_states[handler_num].tx_buffer + b_ind, data, len); b_ind += len;

    crc = crc16(data, len); handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc >> 8);
    handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc & 0xFF);
    handler_states[handler_num].tx_buffer[b_ind++] = 0xBB;

    if (handler_states[handler_num].send_func)
    {
        handler_states[handler_num].send_func(handler_states[handler_num].tx_buffer, b_ind);
    }
}

```

4.4. 制御コマンド例

4.4.1. CAN ポートを使った制御コマンド例

以下の例では拡張フレーム形式のデータフレームを使用します。モーターID は 0x68 としています。

Mode	ID	DATA	Descriptions
Duty Cycle Mode	00 00 00 68	00 00 4E 20	0.2
	00 00 00 68	FF FF B1 E0	-0.2

Current Loop	00 00 01 68	FF FF F0 60	-4A IQ電流
	00 00 01 68	00 00 0F A0	4A IQ電流
Brake Current Mode	00 00 02 68	FF FF F0 60	-4A ブレーキ電流
	00 00 02 68	00 00 0F A0	4A ブレーキ電流
Velocity Loop	00 00 03 68	00 00 13 88	速度 5000 (ERPM)
	00 00 03 68	FF FF EC 78	速度 -5000 (ERPM)
Position Loop	00 00 04 68	00 5B 8D 80	モーター角度 600°
	00 00 04 68	FF A4 72 80	モーター角度 -600°
Position-Velocity Loop	00 00 06 68	00 98 96 80 03 E8 03 E8	モーター角度 1000° 速度 10000ERPM 加速度 10000ERPM/s
	00 00 06 68	FF 67 69 80 FC 18 FC 18	モーター角度 -1000° 速度 -10000ERPM 加速度 -10000ERPM/s
MIT Velocity Loop	00 00 08 68	00 06 66 7F FF 98 67 FF	Kd=2 速度 6rad/s
	00 00 08 68	00 06 66 7F FF 64 87 FF	Kd=2 速度 -6rad/s
MIT Position Loop	00 00 08 68	01 06 66 BD 70 7F F7 FF	Kp=2, Kd=2, モーター角度 6rad
	00 00 08 68	01 06 66 42 8F 7F F7 FF	Kp=2, Kd=2 モーター角度-6rad
MIT Torque Loop	00 00 08 68	00 00 00 7F FF 7F F8 4B	トルク 2Nm
	00 00 08 68	00 00 00 7F FF 7F F8 97	トルク 4Nm

4.4.2. シリアルポート制御コマンド例

Mode	Serial Port Command	Descriptions
Duty Cycle Mode	AA 05 46 00 00 4E 20 D6 4C BB	0.20 Duty Cycle
	AA 05 46 FF FF B1 E0 88 3F BB	-0.20 Duty Cycle
Brake Current Mode	AA 05 48 00 00 13 88 55 E5 BB	5A ブレーキ電流
	AA 05 48 FF FF EC 78 3D C5 BB	-5A ブレーキ電流
Velocity Loop	AA 05 49 00 00 03 E8 90 61 BB	1000 ERPM
	AA 05 49 FF FF FC 18 F8 41 BB	-1000 ERPM
Position Loop	AA 05 4A 0A BA 95 00 E1 4D BB	180° に回転
	AA 05 4A 05 5D 4A 80 84 93 BB	90° に回転
Position-Velocity Loop	AA 0D 3C 00 02 BF 20 00 00 13 88 00 00 75 30 18 1C BB	目標位置180° 速度 5000ERPM、 加速度 30000ERPM/s
Current Loop	AA 05 47 00 00 13 88 30 1C BB	5 A IQ電流
	AA 05 47 FF FF EC 78 58 3C BB	- 5 A IQ電流
MIT Velocity Loop	AA 15 60 00 00 00 00 00 00 17 70 00 00 00 00 00 00 00 00 00 00 07 D0 93 DA BB	Kd=2 速度 6rad/s
	AA 15 60 00 00 00 00 FF FF E8 90 00 00 00 00 00 00 00 00 00 00 07 D0 87 5C BB	Kd=2 速度 -6rad/s
MIT Position Loop	AA 15 60 00 00 17 70 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 07 D0 E3 A9 BB	Kp=2, Kd=2 モーター角度 6rad
	AA 15 60 FF FF E8 90 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 07 D0 BB 05 BB	Kp=2, Kd=2, モーター角度 -6rad
MIT Torque Loop	AA 15 60 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 00 00 00 00 00 00 B9 A2 BB	トルク 2Nm

	AA 15 60 00 00 00 00 00 00 00 00 00 00 0F A0 00 00 00 00 00 00 00 00 00 58 32 BB	トルク 4Nm
--	---	---------