DENSO
b-CAP 通信仕様書
RC8 用

Version 1.0.5

July 16, 2021

【備考】

【改版履歴】

日付	版数	内容
2013-2-12	1.0.0	初版.
2013-8-20	1.0.1	ANSI-C, Java 版サンプルライブラリの参照追記 サンプルプログラム(変数アクセス, タスク制御, ロボット制御)追記 Slave Mode 指令速度/加速度の限界値の説明追記 b-CAP Tester Raw Packet Mode 追記 b-CAP 関数 ID と CAO インターフェースの対応表追記
2013-10-08		Slv Mode に SlvSendFormat , SlvRecvFormat 追記
2014-10-13	1.0.2	Slave Mode の通信速度制限について追記 Slave Mode のコマンド制限について追記
2015-04-02		サンプルプログラム修正 Slave Mode に電流値取得追記
2017-02-07	1.0.3	Slave Mode の付加軸制御について追記
2017-10-19	1.0.4	Service_Start の引数について追記
2021-10-16	1.0.5	b-CAP Slave Mode サンプルコードの修正. b-CAP Slave Mode 未対応機能について追記.

【対応機器】

機種	バージョン	注意事項
RC8	V1.4.0 以降	

目次

1. はじめに

本仕様書は RC8 用 b-CAP の通信プロトコルを規定するものです.

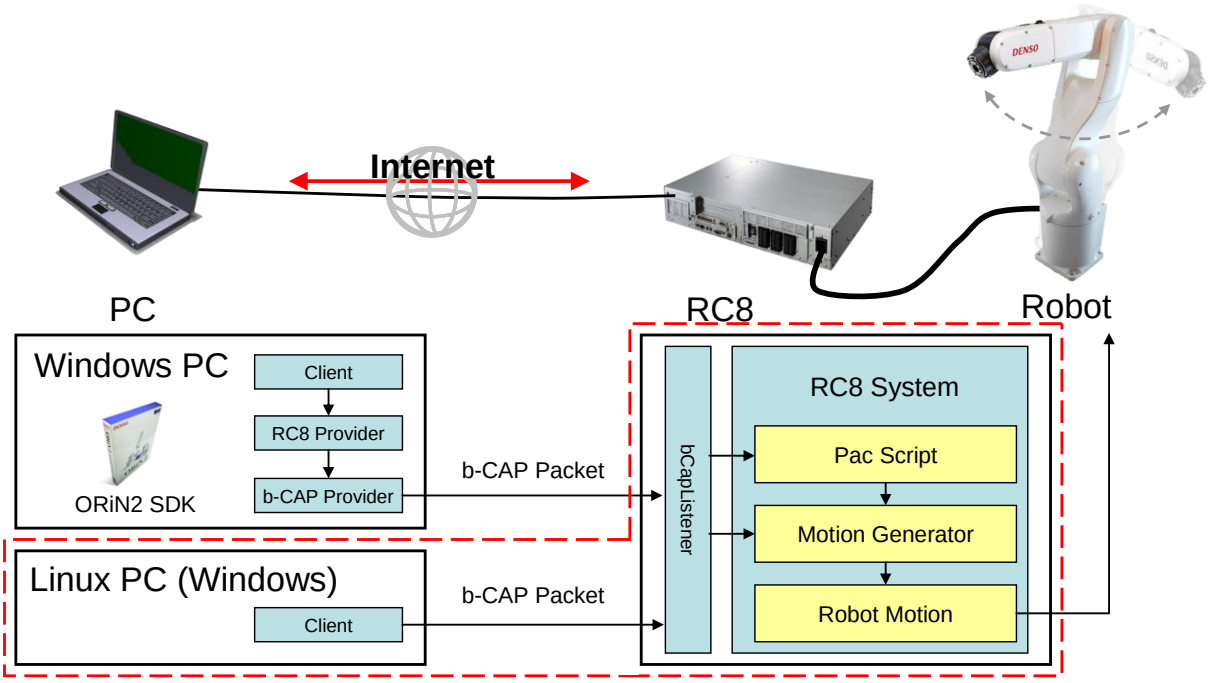
b-CAP は, ORiN で仕様が決められている CAP の概念を踏襲しつつ, 通信速度の向上を狙ったプロトコルです. そのため, b-CAP は CAP ファミリと同様な以下の特徴を持っています.

- CAO プロバイダのオブジェクトモデルと同様なサービス構造
- オブジェクト ID で対象オブジェクトを指定しての関数呼び出し
- サーバからのイベントをポーリングで実現

なお, CAP に関しての詳細な情報は, ORiN SDK に含まれる「CAP プロバイダユーザーズガイド」(CAP_ProvGuide_jp.pdf)に記載されていますので参照して下さい.

1.1. 本書が対象としている環境

本書は, RC8コントローラ Version1.4.0 以降を対象としています. またクライアントソフトウェアが動作する環境としては Linux または Windows を想定しています.



RC8 上では bCapListener が b-CAP パケットを受け取り, パケットの内容に応じて命令を振り分けています. RC8 の内部には命令を言語として解釈する Pac Script, ロボット動作命令が発行された場合に軌道生成を行う Motion Generator, ロボットのリアルタイム制御を行う Robot Motion があります.

ORiN2 SDK がインストールされた Windows 上で動作するクライアントソフトウェアは, RC8 Provider を用いて RC8 を操作することができます. RC8 Provider は, b-CAP Provider を通してコマンドを b-CAP パケットに変換し, RC8 に送信しています.

Linux または ORiN2 SDK がインストールされていない Windows など RC8 Provider を用いることができな

いクライアントは，独自に b-CAP パケットを送信することで，RC8 を操作することができます．

本書では，b-CAP パケットを送受信し，RC8 を操作するための手順について具体例を挙げながら説明します．

1.2. 参考情報

本書は，RC8 の基本的な操作を実現する b-CAP パケットの送信例を記載しています．より詳細な操作を実現する場合には，下記のファイルを参照してください．

b-CAP の基本的な構造は下記を参照してください．

・b-CAP 通信仕様書

ORiN2¥CAP¥b-CAP¥Doc¥b-CAP_Spec_ja.pdf

RC8 用 b-CAP のサポートするコマンドは RC8 Provider に準拠します．コマンドの引数などは下記を参照してください．

・RC8 プロバイダガイド

ORiN2¥CAO¥ProviderLib¥DENSO¥RC8¥Doc¥RC8_ProvGuide_ja.pdf

付属のサンプルライブラリを用いて b-CAP パケットを生成し，コントローラに命令を送ることができます．

・ANSI-C 版サンプルライブラリ

ORiN2¥CAP¥b-CAP¥CapLib¥DENSO¥RC8¥Include¥C

・Java 版サンプルライブラリ

ORiN2¥CAP¥b-CAP¥CapLib¥DENSO¥RC8¥Include¥Java

1.3. b-CAP Slave Mode

Slave Mode は，短い時間間隔で位置・姿勢データを送信することでロボットを操作する機能です．

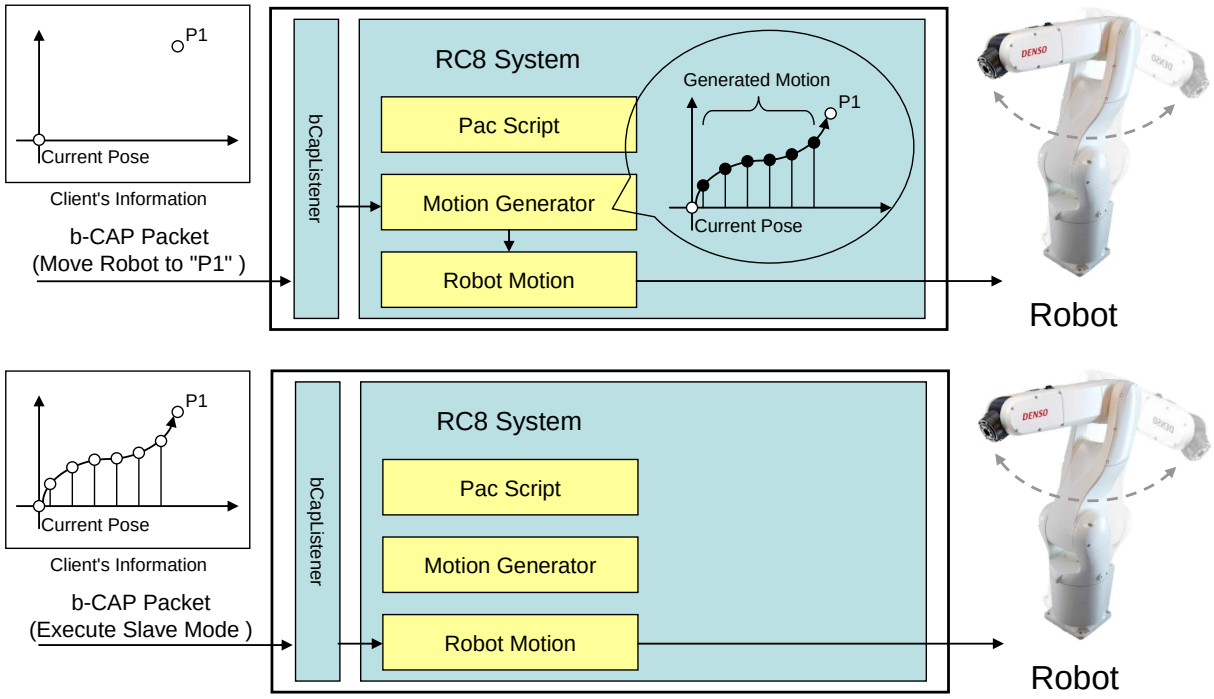
通常のロボット動作命令（例えば"Move 1, "P1""）は，クライアントが指定した目標姿勢を実現するために，RC8 が軌道生成を行い，ロボットのリアルタイム制御を行います．これに対し，Slave Mode では，クライアントがロボットの姿勢を逐次指定することでロボットのリアルタイム制御を行います．この機能を用いることで，クライアントがロボットの軌跡を自由に操作することができます．

Slave Mode は RC8 ロボットコントローラのオプション機能です．ライセンスキーを機能拡張から追加してください．ライセンスキーは，会員サイトの「RC8 無償ライセンス確認」より，RC8 に記載されているシリアルナンバーを入力しご確認ください[1]．

※ライセンスキー確認には，会員サイトへの登録・ログインが必要となります．

http://www.denso-wave.com/ja/robot/index.html

[1] b-CAP Slave ライセンスが有効になっているコントローラは，機能の通信を保護するためその他の通信速度が制限されます．
速度制限によってその他の通信がタイムアウトしてしまう場合は，タイムアウト時間を伸ばすなどの対策を実施してください．


P1
Current Pose
Client's Information
b-CAP Packet
(Move Robot to "P1")
bCapListener
RC8 System
Pac Script
Motion Generator
Robot Motion
Generated Motion
P1
Current Pose
Robot
P1
Current Pose
Client's Information
b-CAP Packet
(Execute Slave Mode)
bCapListener
RC8 System
Pac Script
Motion Generator
Robot Motion
Robot

2. RC8 コントローラのセットアップ

ここでは，b-CAP パケットによりコントローラを操作するために必要な RC8 のセットアップについて説明します．詳細なセットアップ方法については RC8 プロバイダガイドを参照して下さい．

2.1. システムパラメータの設定

b-CAP パケットによってロボットコントローラを操作するには制御対象となるロボットコントローラの通信権と起動権の設定をティーチングペンダント（TP）もしくはミニペンダント（MiniTP）のどちらかで行う必要があります．

通信権は，ロボットコントローラに対してデータの読み込みと書き込みの権限を，通信デバイスに与えるための設定です．変数データの書き込みやロボット制御を行う場合は，書き込みの権限を与えてください．

起動権は，ロボットコントローラに対してプログラムタスクの起動（実行），モータ ON 及びロボット制御（動作指令）の権限を，通信デバイスに与えるための設定です．設定可能な値は，①TP，②I/O，③Ethernet，④Any のいずれかです．Any の場合は通信経路を問わずいつでも起動権が与えられますので，Any に設定する場合はクライアント PC や PLC 側で衝突が起こらないよう，通信デバイス間で排他処理を施してください．

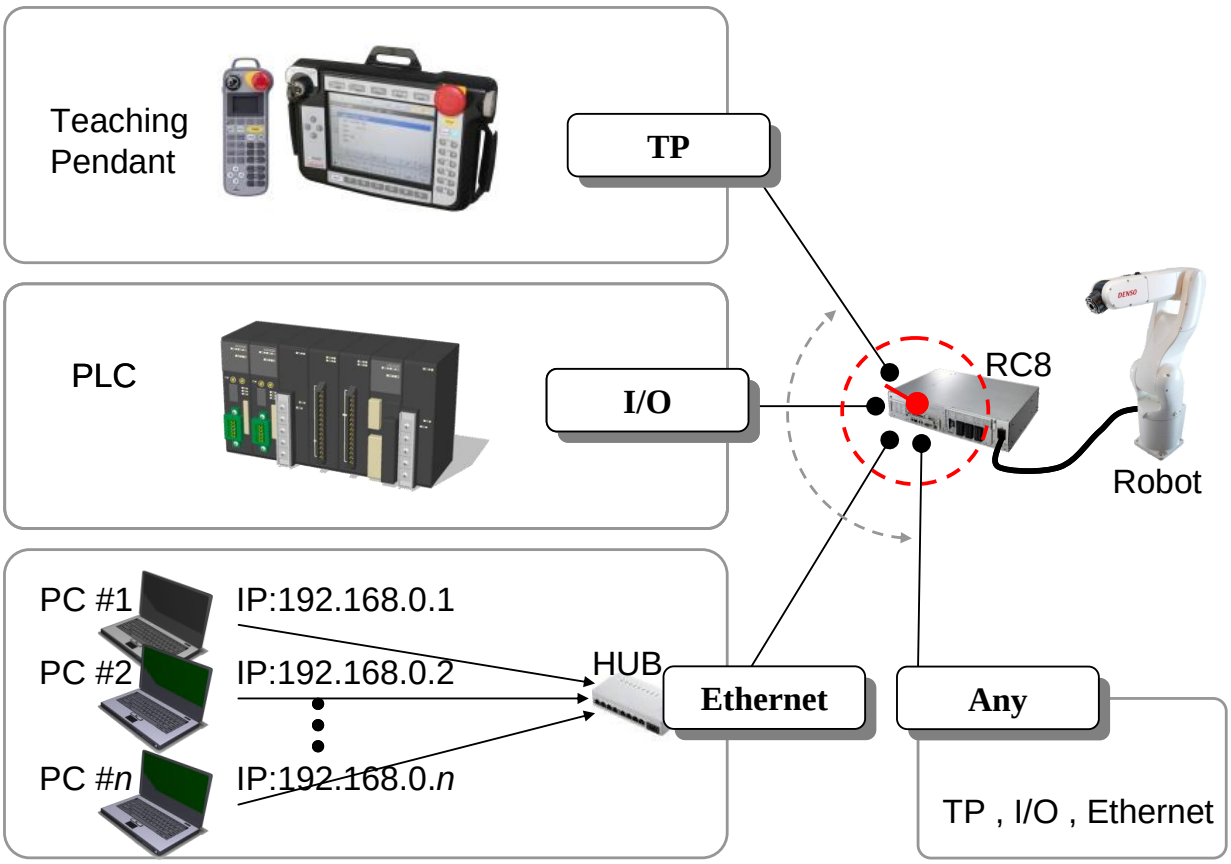


図 2-1　起動権を持つデバイスの設定

接続方法として Ethernet を用いる場合は，クライアント PC の IP アドレスを設定する必要があります．この設

定により，ロボットコントローラは特定のクライアント PC からのみプログラムタスクの起動やロボット制御を可能にします．

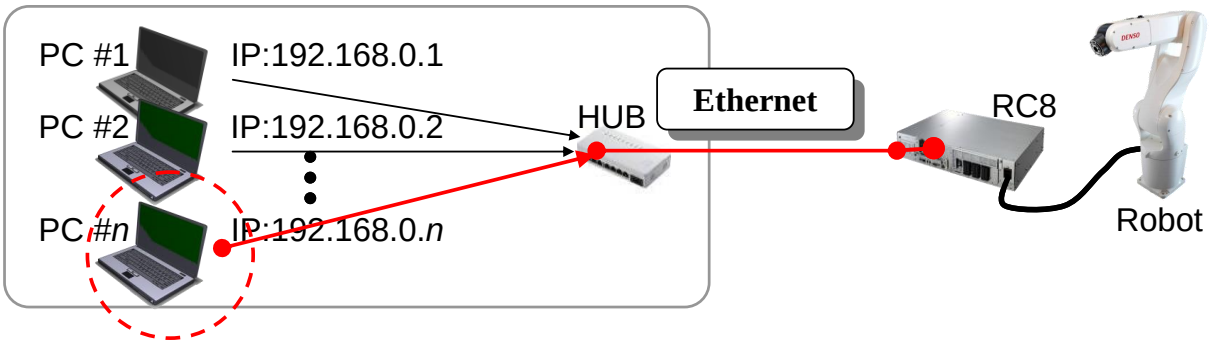


図 2-2 起動権を Ethernet に設定

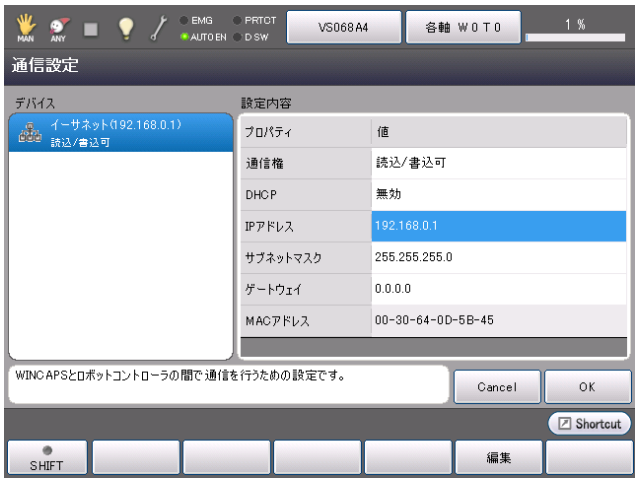


図 2-3 通信権の設定

図 2-4 起動権の設定

2.2. 自動モードの設定

ロボットコントローラに対してプログラムタスクの起動（実行），モータ ON 及びロボット制御（動作指令）を外部のクライアントから行うには，ロボットコントローラが自動モードになっている必要があります．

ロボットコントローラを自動モードにするには，ティーチングペンダントもしくはミニペンダントを図 2-5 の状態にします．

図 2-5 自動モードの設定

3. 通信手順

RC8 用 b-CAP を用いてロボットコントローラと通信を行う手順を，以下に具体例を挙げながら説明します．

3.1. 変数アクセス

変数アクセスを行うには，図 3-1 に示す処理を行います．以降で，各処理の詳細について説明します．

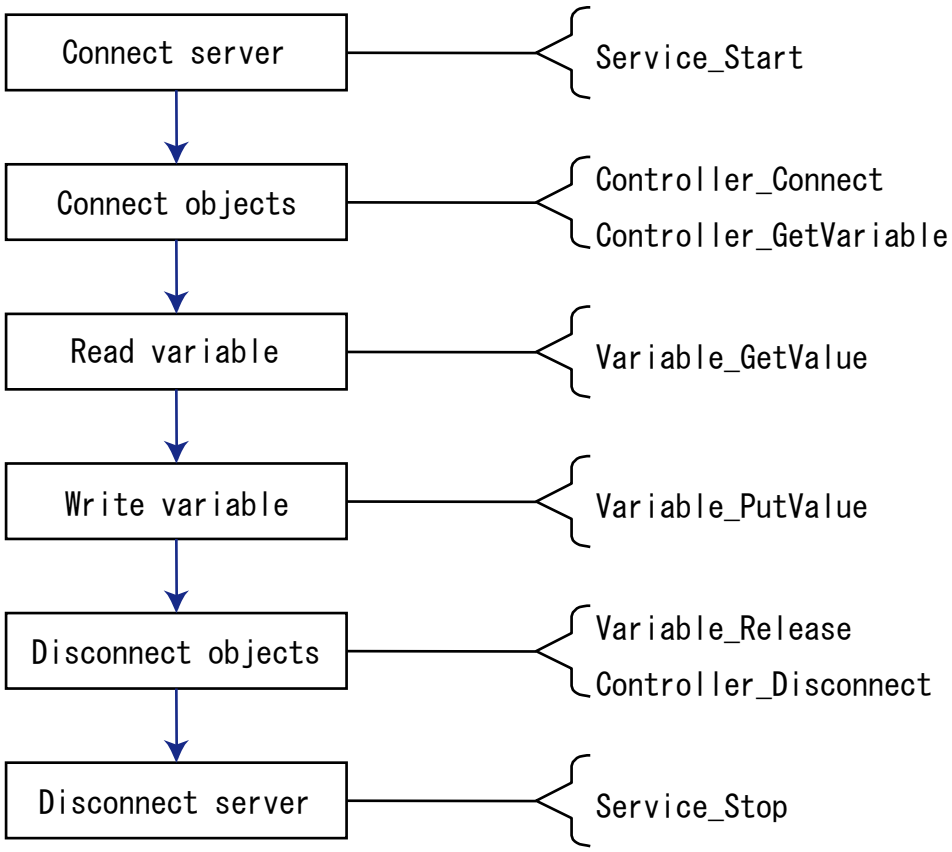


図 3-1　変数アクセスの流れ

3.1.1. サーバへの接続（Connect server）

サーバサービスを開始するには，以下のパケットを送信します．

Service_Start				
送信パケット	クライアント→サーバ: 01 10 00 00 00 00 00 00　00 01 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

受信 パケット	サーバ→クライアント: 01 10 00 00 00 00 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

サーバサービスを開始するパケットには引数としてオプション文字列を指定することもできます．以下にオプション文字列に指定するリストを示します．

表 3-1 Service_Start のオプション文字列

オプション	意味
WDT=<ウォッチドッグタイマ時間>	b-CAP サーバのウォッチドッグタイマ間隔(ms)を指定します． b-CAP サーバでタイムアウト時間より長いコマンドを実行する場合，このオプションを指定することでコマンド完了前にタイムアウトが発生することを防ぎます．b-CAP サーバでコマンドの実行中に指定時間が経過したとき，クライアントに対して特殊パケット（実行中通知パケット）を送信します．クライアントは，サーバから実行中通知パケットを受け取るごとにタイムアウトのカウントをリセットします． この設定には 80ms より大きくタイムアウト時間より小さい値を設定してください．

オプション文字列を指定する場合は以下のパケットを送信します．

Service_Start				
送信 パケット	クライアント→サーバ: 01 2c 00 00 00 01 00 00　00 01 00 00 00 01 00 18 00 00 00 08 00 01 00 00　00 0e 00 00 00 57 00 44 00 54 00 3d 00 34 00 30　00 30 00 04			
	引数	説明	データ型	値
		バイナリ		
	bstrOption	オプション文字列	VT_BSTR	"WDT=400"
		0e 00 00 00 57 00 44 00 54 00 3d 00 34 00 30　00 30 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 00 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.1.2. オブジェクトへの接続 (Connect objects)

各オブジェクトに接続し, オブジェクトのハンドルを取得します. コントローラの変数オブジェクトに接続するには, コントローラハンドルが必要です. コントローラ接続には関数 ID に Controller_Connect(3)を, コントローラの変数オブジェクトに接続するには Controller_GetVariable(9)を用います. 以下にそれぞれのパケットの例を示します. ここでは, IP:192.168.0.1 のコントローラに接続し, システム変数"IO150"のハンドルを取得します. IP は各コントローラで設定した値を用いて下さい.

<table>
<tr><td colspan="4">Controller_Connect</td></tr>
<tr><td colspan="4">コントローラに接続し, コントローラオブジェクトのハンドル(hController)を返します.</td></tr>
<tr><td rowspan="11">送信
パケット</td><td colspan="3">クライアント→サーバ:
01 8A 00 00 00 01 00 00 00 03 00 00 00 04 00 14
00 00 00 08 00 01 00 00 00 0A 00 00 00 62 00 2D
00 43 00 41 00 50 00 2C 00 00 00 08 00 01 00 00
00 22 00 00 00 43 00 61 00 6F 00 50 00 72 00 6F
00 76 00 2E 00 44 00 45 00 4E 00 53 00 4F 00 2E
00 56 00 52 00 43 00 20 00 00 00 08 00 01 00 00
00 16 00 00 00 31 00 39 00 32 00 2E 00 31 00 36
00 38 00 2E 00 30 00 2E 00 31 00 0A 00 00 00 08
00 01 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>bstrCtrlName</td><td>コントローラ名</td><td>VT_BSTR</td><td>"b-CAP"</td></tr>
<tr><td></td><td colspan="3">14
00 00 00 08 00 01 00 00 00 0A 00 00 00 62 00 2D
00 43 00 41 00 50 00</td></tr>
<tr><td>bstrProvName</td><td>プロバイダ名</td><td>VT_BSTR</td><td>"CaoProv.DENSO.VRC"</td></tr>
<tr><td></td><td colspan="3">2C 00 00 00 08 00 01 00 00
00 22 00 00 00 43 00 61 00 6F 00 50 00 72 00 6F
00 76 00 2E 00 44 00 45 00 4E 00 53 00 4F 00 2E
00 56 00 52 00 43 00</td></tr>
<tr><td>bstrPcName</td><td>クライアント PC 名</td><td>VT_BSTR</td><td>"192.168.0.1"</td></tr>
<tr><td></td><td colspan="3">20 00 00 00 08 00 01 00 00
00 16 00 00 00 31 00 39 00 32 00 2E 00 31 00 36
00 38 00 2E 00 30 00 2E 00 31 00</td></tr>
<tr><td>bstrOption</td><td>接続オプション</td><td>VT_BSTR</td><td>空文字列</td></tr>
<tr><td></td><td colspan="3">0A 00 00 00 08
00 01 00 00 00 00 00 00 00</td></tr>
<tr><td rowspan="6">受信
パケット</td><td colspan="3">サーバ→クライアント:
01 1E 00 00 00 01 00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>hController</td><td>コントローラハンドル</td><td>VT_I4</td><td>0x00000002</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 02 00 00 00</td></tr>
</table>

Controller_GetVariable				
コントローラのシステム変数オブジェクトのハンドル(hVariable)を取得します.				
送信パケット	クライアント→サーバ: 01 44 00 00 00 03 00 00 00 09 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 14 00 00 00 08 00 01 00 00 00 0A 00 00 00 49 00 4F 00 31 00 35 00 30 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrName	変数名	VT_BSTR	"IO150"
		14 00 00 00 08 00 01 00 00 00 0A 00 00 00 49 00 4F 00 31 00 35 00 30 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		0A 00 00 00 08 00 01 00 00 00 00 00 00 00		
受信パケット	サーバ→クライアント: 01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		

3.1.3. 変数リード・ライト (Read/Write variable)

接続した変数の値を取得・設定します. 値を取得するには関数 ID に Variable_GetValue(101)を, 値を設定するには Variable_PutValue(102)を用います. 以下にそれぞれのパケットの例を示します.

Variable_GetValue				
"hVariable"で指定された変数オブジェクトの値を返します.				
送信パケット	クライアント→サーバ: 01 1E 00 00 00 04 00 00 00 65 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		

受信パケット	サーバ→クライアント: 01 1C 00 00 00 04 00 00　00 00 00 00 00 01 00 08 00 00 00 0B 00 01 00 00　00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	pVal	変数"IO150"の値	VT_BOOL	0x0000 (FALSE)
		00 08 00 00 00 0B 00 01 00 00　00 00 00		

Variable_PutValue				
"hVariable"で指定された変数オブジェクトの値を書き込みます.				
送信パケット	クライアント→サーバ: 01 2A 00 00 00 05 00 00　00 66 00 00 00 02 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 08 00 00 00 0B 00 01 00 00 00 FF　FF 04			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00　00 03 00 00 00		
	newVal	新しく書き込む値	VT_BOOL	0xFFFF (TRUE)
		08 00 00 00 0B 00 01 00 00 00 FF　FF		
受信パケット	サーバ→クライアント: 01 10 00 00 00 05 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.1.4. オブジェクトとの切断 (Disconnect objects)

接続したオブジェクトを切断します. 変数オブジェクトの切断には関数 ID に Variable_Release(111)を, コントローラオブジェクトの切断には Controller_Disconnect(4)を用います. 以下にそれぞれのパケットの例を示します.

Variable_Release				
クライアントを変数ハンドル"hVariable"で指定した変数オブジェクトから切断します.				
送信パケット	クライアント→サーバ: 01 1E 00 00 00 06 00 00　00 6F 00 00 00 01 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		

	hVariable	変数ハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
受信パケット	サーバ→クライアント: 01 10 00 00 00 06 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

Controller_Disconnect				
クライアントをコントローラハンドル"hController"で指定したコントローラオブジェクトから切断します.				
送信パケット	クライアント→サーバ: 01 1E 00 00 00 07 00 00 00 04 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x00000002
		00 00 00 03 00 01 00 00 00 02 00 00 00 0A		
受信パケット	サーバ→クライアント: 01 10 00 00 00 07 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.1.5. サーバとの切断 (Disconnect server)

サーバサービスを停止するには, 以下のパケットを送信します.

Service_Stop				
送信パケット	クライアント→サーバ: 01 10 00 00 00 08 00 00 00 02 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		
受信パケット	サーバ→クライアント: 01 10 00 00 00 08 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		

	なし	-	-	-
		–		

3.1.6. その他の変数へのアクセス

RC8 には I 型変数や IO 型変数，システム変数などさまざまな変数があります．RC8 が提供している変数は「RC8 プロバイダガイド 5.3.変数一覧」で確認することができます．ここでは，RC8 プロバイダガイドに記載されている変数アクセス方法を b-CAP ではどのように表現するか具体例を挙げて示します．

RC8 プロバイダガイドでは，コントローラクラスのシステム変数"@MODE"にアクセスする場合，以下のように記載されます．

```
Dim caoVar as CaoVariable
Set caoVar = caoCtrl.AddVariable("@MODE","")　‘システム変数@MODE を指定
```

これを b-CAP で表現すると以下のようなパケットになります．

送信パケット	クライアント→サーバ: 01 44 00 00 00 02 00 00　00 09 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 02 00 00 00 14 00 00 00 08 00 01 00 00 00 0A　00 00 00 40 00 4D 00 4F 00 44 00 45 00 0A 00 00　00 08 00 01 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x00000002
		00 00 00 03 00 01 00 00　00 02 00 00 0A		
	bstrName	変数名	VT_BSTR	"@MODE"
		00 08 00 01 00 00 00 0A　00 00 00 40 00 4D 00 4F 00 44 00 45 00　14 00 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		00 00 00　0A 00 00　00 08 00 01 00 00 00 00		
受信パケット	サーバ→クライアント: 01 1E 00 00 00 03 00 00　00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00　00 03 00 00 0A		

b-CAP の関数 ID が RC8 Provider のメソッドに対応しており，上記の場合 Controller_GetVariable(9)が caoCtrl.AddVariable に対応しています．caoCtrl や caoVar などの CAO クラスオブジェクトは b-CAP のオブジェクトハンドルに対応しており，上記の場合 caoCtrl はコントローラハンドル，caoVar は変数ハンドルに対応しています．

3.1.7. サンプルプログラム

以下に，ANSI-C 版サンプルライブラリを利用した変数アクセスのサンプルプログラムを示します．

サンプルプログラムは変数 IO150（I/O の 150 番目）の読み書きを行います．IP は各コントローラで設定した値を用いてください．サンプルプログラムでは以下の設定値を用いて接続しています．

IP:192.168.0.1

List 3-1 bCapVariable.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM            5007


int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hVar;
	HRESULT hr;

	/* Open socket */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
	bstrName = SysAllocString(L"");
	bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
	bstrMachine = SysAllocString(L"localhost");
	bstrOpt = SysAllocString(L"");
	hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
	SysFreeString(bstrName);
	SysFreeString(bstrProv);
	SysFreeString(bstrMachine);
	SysFreeString(bstrOpt);
	if FAILED(hr) return (hr);

	/* Get variable handle */
	BSTR bstrVarName, bstrVarOpt;
	bstrVarName = SysAllocString(L"IO150");
	bstrVarOpt = SysAllocString(L"");
	hr = bCap_ControllerGetVariable(fd, hCtrl, bstrVarName, bstrVarOpt, &hVar);
	SysFreeString(bstrVarName);
```

```
        SysFreeString(bstrVarOpt);
        if FAILED(hr) return (hr);

        /* Read variable */
        bCap_VariableGetValue(fd, hVar, &vntResult);

        /* Write variable */
        vntResult.lVal = -1L;
        vntResult.vt = VT_I4;
        bCap_VariablePutValue(fd, hVar, vntResult);

        /* Release variable handle */
        bCap_VariableRelease(fd, &hVar);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

3.2. タスク制御

タスク制御を行うには，図 3-2に示す処理を行います．タスクを起動するには，コントローラが自動モードになっている必要があります．またコントローラの起動権の設定をクライアントPCのIPに設定してください．詳しくは「2.RC8コントローラのセットアップ」を参照してください．以降で，各処理の詳細を説明します．

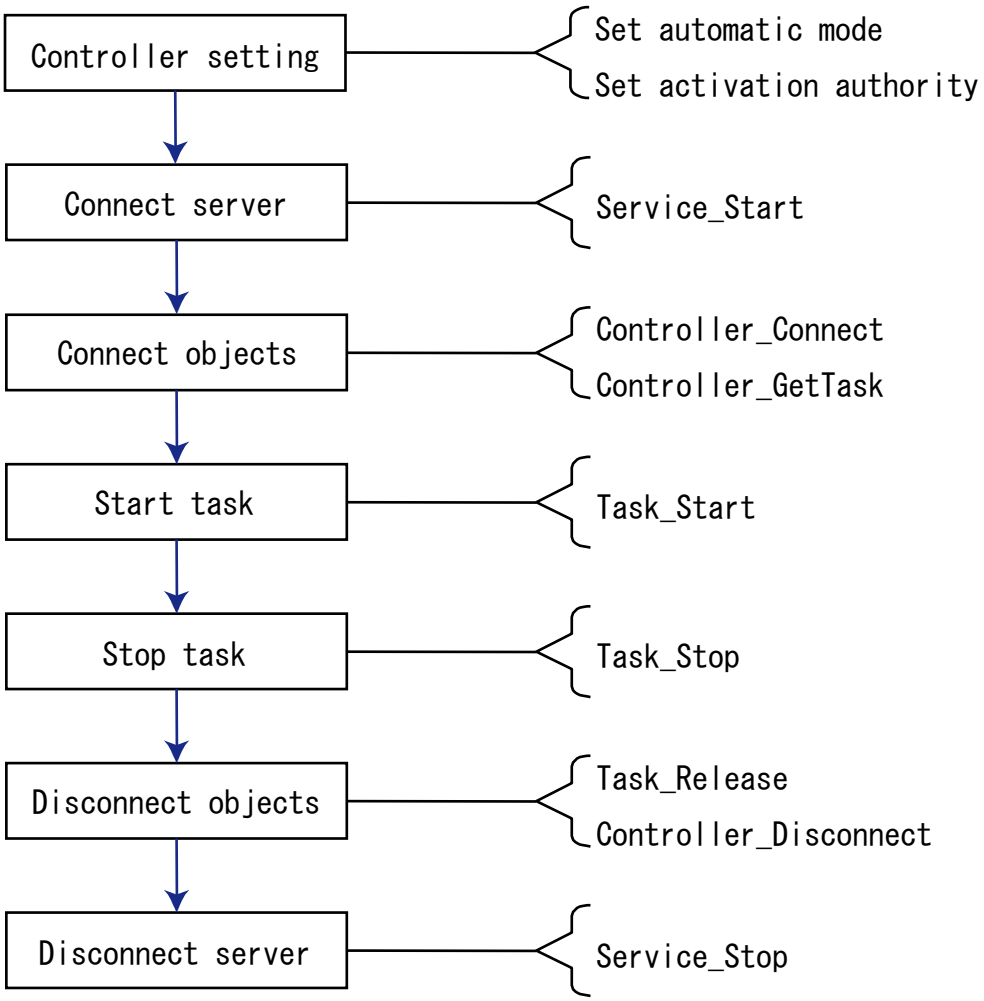


図 3-2 タスク制御の流れ

3.2.1. オブジェクトへの接続 (Connect objects)

コントローラオブジェクトへ接続するまでの手順は「3.1 変数アクセス」を参照して下さい. タスクオブジェクトに接続するには関数 ID に Controller_GetTask(8)を用います. また引数として, コントローラハンドルが必要です. 以下にタスク"Pro1"のハンドルを取得するパケットを示します.

<table>
<tr><th colspan="5">Controller_GetTask</th></tr>
<tr><td colspan="5">タスクオブジェクトのハンドル(hTask)を取得します.</td></tr>
<tr><td rowspan="2">送信
パケット</td><td colspan="4">クライアント→サーバ:
01 42 00 00 00 03 00 00 00 08 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 12 00 00
00 08 00 01 00 00 00 08 00 00 00 50 00 72 00 6F
00 31 00 0A 00 00 00 08 00 01 00 00 00 00 00 00
00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
</table>

		バイナリ		
	hController	コントローラハンドル	VT_I4	0x00000002
		00 00 00 03 00 01 00 00 00 02 00 00 00 0A		
	bstrName	タスク名	VT_BSTR	"Pro1"
		00 08 00 01 00 00 00 08 00 00 00 50 00 72 00 6F 00 31 00 12 00 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		00 08 00 01 00 00 00 00 00 00 00 0A 00 00 00		
受信パケット	サーバ→クライアント: 01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hTask	タスクハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		

3.2.2. タスクの開始・停止 (Start/Stop task)

接続したタスクの開始・停止を行います. タスクを開始するには関数 ID に Task_Start(88)を, 停止するには Task_Stop(89)を用います. 以下にそれぞれのパケットの例を示します.

Task_Start				
タスクをサイクル実行で開始します.				
送信パケット	クライアント→サーバ: 01 3A 00 00 00 04 00 00 00 58 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hTask	タスクハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
	lMode	開始モード(2:サイクル実行)	VT_I4	0x00000002
		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		
受信パケット	サーバ→クライアント: 01 10 00 00 00 04 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値

		バイナリ		
	なし	-	-	-
		-		

Task_Stop				
タスクをサイクル停止で停止します.				
送信 パケット	クライアント→サーバ: 01 3A 00 00 00 05 00 00　00 59 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 03　00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00　00 04			
	引数	説明	データ型	値
		バイナリ		
	hTask	タスクハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00　00 03 00 00 00 0A		
	lMode	停止モード(3:サイクル停止)	VT_I4	0x00000003
		00 03 00 01 00 00 00 03　00 00 00 0A 00 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		00 01 00 00 00 00 00 00　00 0A 00 00 00 08		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 05 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.2.3. オブジェクトとの切断 (Disconnect objects)

接続したオブジェクトから切断します. タスクオブジェクト切断以降の手順は「3.1 変数アクセス」を参照して下さい. タスクオブジェクトを切断するには関数 ID に Task_Release(99)を用います. 以下にタスクオブジェクトを切断するパケットの例を示します.

Task_Release				
クライアントをタスクハンドル"hTask"で指定したタスクオブジェクトから切断します.				
送信 パケット	クライアント→サーバ: 01 1E 00 00 00 06 00 00　00 63 00 00 00 01 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hTask	タスクハンドル	VT_I4	0x00000003

		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
受信パケット	サーバ→クライアント: 01 10 00 00 00 06 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.2.4. サンプルプログラム

以下に, ANSI-C 版サンプルライブラリを利用したタスク制御のサンプルプログラムを示します.

サンプルプログラムはタスク"Pro1"の制御(連続実行とサイクル停止)を行います.

List 3-2 bCapTask.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM           5007

int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hTask;
	int32_t lMode;
	HRESULT hr;

	/* Init and Start b-CAP */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
	bstrName = SysAllocString(L"");
	bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
	bstrMachine = SysAllocString(L"localhost");
	bstrOpt = SysAllocString(L"");
	hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
	SysFreeString(bstrName);
	SysFreeString(bstrProv);
	SysFreeString(bstrMachine);
	SysFreeString(bstrOpt);
	if FAILED(hr) return (hr);

	/* Get task handle */
	BSTR bstrTskName, bstrTskOpt;
	bstrTskName = SysAllocString(L"Pro1");
	bstrTskOpt = SysAllocString(L"");
	hr = bCap_ControllerGetTask(fd, hCtrl, bstrTskName, bstrTskOpt, &hTask);
	SysFreeString(bstrTskName);
```

```
        SysFreeString(bstrTskOpt);
        if FAILED(hr) return (hr);

        /* Start task */
        lMode = 2L;
        bCap_TaskStart(fd, hTask, lMode, bstrTskOpt);

        /* Stop task */
        lMode = 3L;
        bCap_TaskStop(fd, hTask, lMode, bstrTskOpt);

        /* Release task handle */
        bCap_TaskRelease(fd, &hTask);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

3.3. ロボット制御

ロボット制御を行うには，図 3-3に示す処理を行います．ロボット動作を行うには，コントローラが自動モードになっている必要があります．またコントローラの起動権の設定をクライアントPCのIPに設定してください．詳しくは「2.RC8コントローラのセットアップ」を参照してください．以降で，各処理の詳細について説明します．

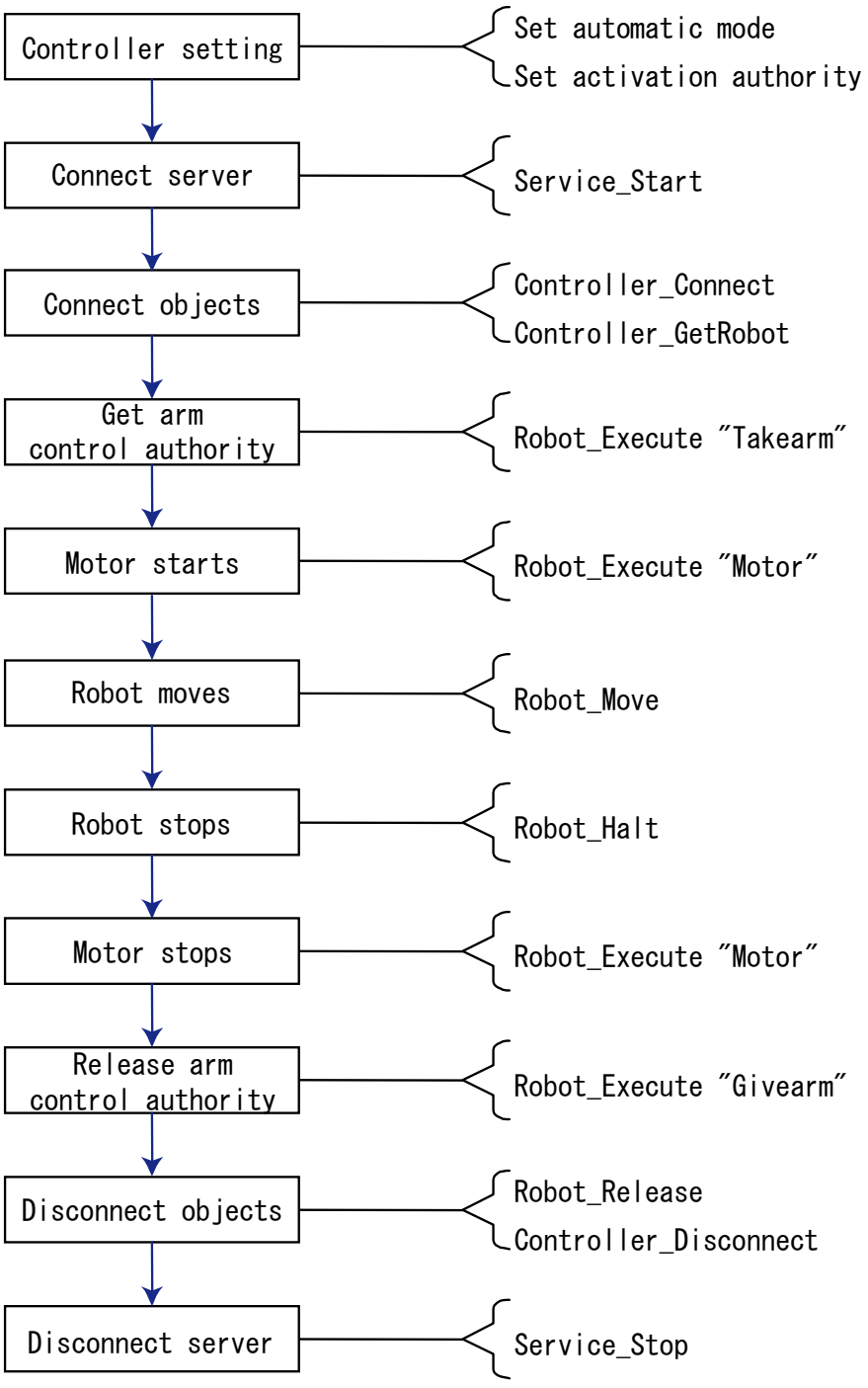


図 3-3 ロボット制御の流れ

3.3.1. オブジェクトへの接続 (Connect objects)

コントローラオブジェクトへ接続するまでの手順は「3.1 変数アクセス」を参照して下さい. ロボットオブジェクトに接続するには関数 ID として Controller_GetRobot(7)を用います. また引数として, コントローラハンドルが必要です. 以下にロボットオブジェクトに接続するパケットの例を示します.

Controller_GetRobot				
ロボットオブジェクトのハンドル(hRobot)を取得します.				
送信 パケット	クライアント→サーバ: 01 40 00 00 00 02 00 00　00 07 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 02 00 00 00 10 00 00 00 08 00 01 00 00 00 06　00 00 00 41 00 72 00 6D 00 0A 00 00 00 08 00 01　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00　00 02 00 00 00		
	bstrName	ロボット名	VT_BSTR	"Arm"
		10 00 00 00 08 00 01 00 00 00 06　00 00 00 41 00 72 00 6D 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		0A 00 00 00 08 00 01　00 00 00 00 00 00 00		
受信 パケット	サーバ→クライアント: 01 1E 00 00 00 03 00 00　00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00　00 03 00 00 00		

3.3.2. アーム制御権の取得と解放 (Get/Release arm control authority)

ロボット制御を行うには, ロボットのアーム制御権を取得する必要があります. また, コントローラと切断する前にロボットのアーム制御権を解放する必要があります. それぞれRobot_Execute(64)のコマンドとして実装されています. 以下にそれぞれのパケットの例を示します.

Robot_Execute "Takearm", (0, 1)				
ロボットのアーム制御権を取得します.				
送信 パケット	クライアント→サーバ: 01 4C 00 00 00 05 00 00　00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 18 00 00 00 08 00 01 00 00 00 0E　00 00 00 54 00 61 00 6B 00 65 00 61 00 72 00 6D　00 0E 00 00 00 03 20 02 00 00 00 00 00 00 00 01　00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00　00 03 00 00 00		

	bstrCommand	コマンド	VT_BSTR	"Takearm"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 54 00 61 00 6B 00 65 00 61 00 72 00 6D 00		
	vntParam	パラメータ	VT_I4 | VT_ARRAY	0, 1
		0E 00 00 00 03 20 02 00 00 00 00 00 00 00 01 00 00 00		
受信パケット	サーバ→クライアント: 01 1A 00 00 00 05 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

Robot_Execute "Givearm"				
ロボットのアーム制御権を解放します.				
送信パケット	クライアント→サーバ: 01 44 00 00 00 0A 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 18 00 00 00 08 00 01 00 00 00 0E 00 00 00 47 00 69 00 76 00 65 00 61 00 72 00 6D 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	コマンド	VT_BSTR	"Givearm"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 47 00 69 00 76 00 65 00 61 00 72 00 6D 00		
	vntParam	パラメータ	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		
受信パケット	サーバ→クライアント: 01 1A 00 00 00 0A 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

3.3.3. モータの起動と停止 (Motor starts/stops)

ロボット制御を行うには，ロボットのモータが起動している必要があります．モータ制御は Robot_Execute (64)のコマンドとして実装されています．以下にモータの起動および停止パケットの例を示します．

<table>
<tr><th colspan="5">Robot_Execute "Motor", (1, 0)</th></tr>
<tr><td colspan="5">ロボットのモータを起動します．</td></tr>
<tr><td rowspan="8">送信
パケット</td><td colspan="4">クライアント→サーバ:
01 48 00 00 00 06 00 00　00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00　00 03 00 00 00 14 00 00
00 08 00 01 00 00 00 0A　00 00 00 4D 00 6F 00 74
00 6F 00 72 00 0E 00 00　00 03 20 02 00 00 00 01
00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>hRobot</td><td>ロボットハンドル</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00　00 03 00 00 00 0A</td></tr>
<tr><td>bstrCommand</td><td>コマンド</td><td>VT_BSTR</td><td>"Motor"</td></tr>
<tr><td colspan="3">00 08 00 01 00 00 00 0A　00 00 00 4D 00 6F 00 74
00 6F 00 72 00 0E 00 00</td></tr>
<tr><td>vntParam</td><td>パラメータ</td><td>VT_I4 | VT_ARRAY</td><td>1, 0</td></tr>
<tr><td colspan="3">00 03 20 02 00 00 00 01
00 00 00 00 00 00 00 00</td></tr>
<tr><td rowspan="6">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 1A 00 00 00 06 00 00　00 00 00 00 00 01 00 06
00 00 00 00 00 01 00 00　00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>vntReturn</td><td>戻り値</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td colspan="3">00 00 00 00 00 01 00 00　00</td></tr>
</table>

<table>
<tr><th colspan="5">Robot_Execute "Motor", (0, 0)</th></tr>
<tr><td colspan="5">ロボットのモータを停止します．</td></tr>
<tr><td rowspan="6">送信
パケット</td><td colspan="4">クライアント→サーバ:
01 48 00 00 00 09 00 00　00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00　00 03 00 00 00 14 00 00
00 08 00 01 00 00 00 0A　00 00 00 4D 00 6F 00 74
00 6F 00 72 00 0E 00 00　00 03 20 02 00 00 00 00
00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>hRobot</td><td>ロボットハンドル</td><td>VT_I4</td><td>0x00000003</td></tr>
</table>

				0A
		00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	コマンド	VT_BSTR	"Motor"
				14 00 00
		00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00		
	vntParam	パラメータ	VT_I4 | VT_ARRAY	0, 0
				0E 00 00 00 03 20 02 00 00 00 00
		00 00 00 00 00 00 00		
受信パケット	サーバ→クライアント: 01 1A 00 00 00 09 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
				06
		00 00 00 00 00 01 00 00 00		

3.3.4. ロボットの移動と停止 (Robot moves/halts)

ロボットを動作させるには関数IDにRobot_Move(72)を用います. Moveコマンドの詳細はRC8プロバイダガイドを参照してください. Move時に"NEXT"オプションをつけた場合, 関数IDにRobot_Halt(70)を用いることでロボットの動作を停止させることができます. 以下にそれぞれのパケットの例を示します. ここでは, ロボットをP1(P型の1番目)に格納された場所へ"NEXT"オプションを付けて移動させます.

Robot_Move 1, "P1", "NEXT"				
動作コマンド"MOVE 1, "P1" "NEXT""を実行します.				
送信パケット	クライアント→サーバ: 01 54 00 00 00 07 00 00 00 48 00 00 00 04 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 0E 00 00 00 08 00 01 00 00 00 04 00 00 00 50 00 31 00 12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
				0A
		00 00 00 03 00 01 00 00 00 03 00 00 00		
	lComp	補間指定	VT_I4	0x00000001
				0A 00 00
		00 03 00 01 00 00 00 01 00 00 00		
	vntPose	ポーズ列	VT_BSTR	"P1"
				0E 00 00 00 08
		00 01 00 00 00 04 00 00 00 50 00 31 00		
	bstrOption	動作オプション	VT_BSTR	"NEXT"

		12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 07 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

Robot_Halt			
ロボットの動作を停止します.			
送信 パケット	クライアント→サーバ: 01 2C 00 00 00 08 00 00 00 46 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04		
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrOption	オプション文字列	VT_BSTR	空文字列
		0A 00 00 00 08 00 01 00 00 00 00 00 00 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 08 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.3.5. オブジェクトとの切断 (Disconnect objects)

接続したオブジェクトから切断します. ロボットオブジェクト切断以降の手順は「3.1 変数アクセス」を参照して下さい. ロボットオブジェクトを切断するには関数 ID に Robot_Release(84)を用います. 以下にロボットオブジェクトを切断するパケットの例を示します.

Robot_Release			
クライアントをロボットハンドル"hRobot"で指定したロボットオブジェクトから切断します.			
送信 パケット	クライアント→サーバ: 01 1E 00 00 00 0B 00 00 00 54 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04		
	引数	説明	データ型	値

		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00		0A
受信パケット	サーバ→クライアント: 01 10 00 00 00 0B 00 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

3.3.6. その他の Execute メソッド

RC8 は各 CAO クラスオブジェクトに固有の拡張コマンドを有しています. RC8 が提供している拡張コマンドは「RC8 プロバイダガイド 5.2.11.CaoController::Execute メソッド」などで確認することができます. ここでは, RC8 プロバイダガイドに記載されている拡張コマンドの実行方法を b-CAP ではどのように表現するか具体例を挙げて示します.

RC8 プロバイダガイドでは, コントローラクラスの Execute メソッド"ClearError"を実行する場合, 以下のように記載されます.

```
caoCtrl.Execute "ClearError"
```

これを b-CAP で表現すると以下のようなパケットになります.

送信パケット	クライアント→サーバ 01 4A 00 00 00 12 00 00 00 11 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 1E 00 00 00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x0000002
		00 00 00 03 00 01 00 00 00 02 00 00 00		0A
	bstrCommand	コマンド文字列	VT_BSTR	"ClearError"
		00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00		1E 00 00
	vntParam	パラメータ	VT_EMPTY	
		00 00 00 00 00 01 00 00 00		06

<table>
<tr><td rowspan="5">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 1A 00 00 00 12 00 00 00 00 00 00 00 00 01 00 06
00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>vntReturn</td><td>戻り値</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td colspan="3">00 00 00 00 00 01 00 00 00 06</td></tr>
</table>

b-CAP の関数 ID が RC8 Provider のメソッドに対応しており，上記の場合 Controller_Execute(17)が caoCtrl.Execute に対応しています．caoCtrl などの CAO クラスオブジェクトは b-CAP のオブジェクトハンドルに対応しており，上記の場合 caoCtrl はコントローラハンドルに対応しています．"ClearError"などの拡張コマンド名は文字列の情報をバイト配列化したものをオブジェクトハンドルの後に追加することで b-CAP のパケットにすることができます．実行する拡張コマンドに引数が必要な場合は，引数の情報をバイト配列化したものをコマンド文字列の後に追加してください．

3.3.7. サンプルプログラム

以下に，ANSI-C 版サンプルライブラリを利用したロボット制御のサンプルプログラムを示します．

サンプルプログラムはロボットを P1(P 型の 1 番目)に格納された場所へ移動させます．

List 3-3 bCapRobot.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM           5007

int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hRobot;
	HRESULT hr;

	/* Init socket */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
	bstrName = SysAllocString(L"");
	bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
	bstrMachine = SysAllocString(L"localhost");
	bstrOpt = SysAllocString(L"");
```

```
/* Connect controller */
hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
SysFreeString(bstrName);
SysFreeString(bstrProv);
SysFreeString(bstrMachine);
SysFreeString(bstrOpt);
if FAILED(hr) return (hr);

/* Get robot handle */
BSTR bstrRobotName, bstrRobotOpt;
bstrRobotName = SysAllocString(L"Arm");
bstrRobotOpt = SysAllocString(L"");
hr = bCap_ControllerGetRobot(fd, hCtrl, bstrRobotName, bstrRobotOpt, &hRobot);
SysFreeString(bstrRobotName);
SysFreeString(bstrRobotOpt);
if FAILED(hr) return (hr);

/* Get arm control authority */
BSTR bstrCommand;
VARIANT vntParam;
bstrCommand = SysAllocString(L"Takearm");
vntParam.bstrVal = SysAllocString(L"");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Motor on */
bstrCommand = SysAllocString(L"Motor");
vntParam.bstrVal = SysAllocString(L"1");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Move to P1 */
VARIANT vntPose;
BSTR bstrMoveOpt;
vntPose.bstrVal = SysAllocString(L"P1");
vntPose.vt = VT_BSTR;
bstrMoveOpt = SysAllocString(L"");
hr = bCap_RobotMove(fd, hRobot, 1L, vntPose, bstrMoveOpt);
VariantClear(&vntPose);
SysFreeString(bstrMoveOpt);
if FAILED(hr) return (hr);

/* Motor off */
bstrCommand = SysAllocString(L"Motor");
vntParam.bstrVal = SysAllocString(L"0");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Release arm control authority */
bstrCommand = SysAllocString(L"Givearm");
vntParam.bstrVal = SysAllocString(L"");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);
```

```
        /* Release robot handle */
        bCap_RobotRelease(fd, &hRobot);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

4. b-CAP Slave Mode の使い方

4.1. Slave Mode とは

Slave Mode は短い時間間隔で位置・姿勢データを送信することでロボットをコントロールする機能です.

b-CAP には Slave Mode として以下の 5 つの関数が実装されています.

・slvChangeMode[2]: Slave Mode の設定を変更します.
・slvGetMode: 現在の Slave Mode の設定を取得します.
・slvMove: 指定した位置・姿勢にロボットを移動させます.
・slvSendFormat: slvMove コマンドのパラメータフォーマットを変更します.
・slvRecvFormat: slvMove コマンドの戻り値フォーマットを変更します.

4.2. Slave Mode 関数

Slave Mode 関数は, Robot_Execute のコマンドとして実装されています.

関数	Robot_Execute		
関数 ID	64		
引数	VT_I4	hRobot	ロボットハンドル
	VT_BSTR	bstrCommand	コマンド名
	VARIANT	vntParam	パラメータ
戻り値	VARIANT	pVal	結果値
説明	ロボット(hRobot)のコマンドを実行します.		

"bstrCommand"に下記のコマンド名を入れることにより, コマンドを実行することができます.

表 4-1 Slave Mode 関数一覧

コマンド名	パラメータ			戻り値	動作
slvChangeMode	<スレーブモード:VT_I4>			なし	Slave Mode の設定を変更します. Slave Mode に切り替える時には, ロボットが停止状態である必要があります. また, Slave Mode に切り替えるクライア
	値	位置形式	動作		
	0x000	-	モード解除		
	0x001	P型	モード0設定		
	0x002	J型	モード0設定		
	0x003	T型	モード0設定		
	0x101	P型	モード1設定		
	0x102	J型	モード1設定		

[2] Slave Mode 中は slvGetMode または slvMove のコマンドのみ使用できます.

<table>
<tr><td></td><td>0x103 T型 モード1設定
0x201 P型 モード2設定
0x202 J型 モード2設定
0x203 T型 モード2設定</td><td></td><td>ントがアーム制御権を取得している必要があります.</td></tr>
<tr><td>slvGetMode</td><td>なし</td><td><スレーブモード:VT_I4>
slvChangeMode のパラメータを参照して下さい.</td><td>現在の Slave Mode の設定を取得します.</td></tr>
<tr><td>slvMove</td><td>slvSendFormat および slvRecvFormat の設定によって異なります. 表 4-2 を参照してください.</td><td>slvRecvFormat の設定によって戻り値が異なります.
表 4-3 を参照してください.</td><td>指定した位置・姿勢にロボットを移動させます.</td></tr>
<tr><td>slvSendFormat</td><td><拡張フォーマット:VT_I4>
<table><tr><td>値</td><td>拡張</td></tr><tr><td>0x0000</td><td>なし</td></tr><tr><td>0x0020</td><td>HandIO 状態</td></tr><tr><td>0x0100</td><td>MiniIO 状態</td></tr><tr><td>0x0120</td><td>MiniIO+HandIO 状態</td></tr><tr><td>0x0200</td><td>汎用 IO 状態</td></tr><tr><td>0x0220</td><td>汎用 IO+HandIO 状態</td></tr></table></td><td>なし</td><td>MiniIO , HandIO , 汎用 IO から信号を出力できるように設定できます. [3]</td></tr>
<tr><td>slvRecvFormat</td><td><戻り値フォーマット:VT_I4 | VT_ARRAY>
<第 1 引数>
<table><tr><td>値</td><td>フォーマット</td></tr><tr><td>0x0001</td><td>P 型</td></tr><tr><td>0x0002</td><td>J 型</td></tr><tr><td>0x0003</td><td>T 型</td></tr><tr><td>0x0004</td><td>P 型+J 型</td></tr><tr><td>0x0005</td><td>T 型+J 型</td></tr></table></td><td>なし</td><td>現在のロボット位置とタイムスタンプと MiniIO , HandIO , 汎用 IO の状態を取得します. 3</td></tr>
</table>

[3] 指定サイズが大きくなると通信遅延が原因で正常に動作しない場合があります.

	0x0010 タイムスタンプ 0x0020 HandIO 状態 0x0040 電流値取得 0x0100 MiniIO 状態 0x0200 汎用 IO 状態 <第 2 引数> 0:タイムスタンプを ms で返す(default) 1:タイムスタンプを us で返す ※省略時は 0 になります.		

表 4-2 slvMove コマンドパラメータフォーマット一覧

拡張フォーマット	パラメータの型	パラメータ
位置のみ(default)	VT_R8 | VT_ARRAY	位置(VT_R8 | VT_ARRAY)
位置と受信汎用 IO のパラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 受信汎用 IO のオフセット(VT_I4) 受信汎用 IO のサイズ(VT_I4)
位置と HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) HandIO 状態(VT_I4)
位置と HandIO 状態と受信汎用 IO のパラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 受信汎用 IO のオフセット(VT_I4) 受信汎用 IO のサイズ(VT_I4) HandIO 状態(VT_I4)
位置と MiniIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4)
位置と MiniIO 状態と受信汎用 IO のパラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) 受信汎用 IO のオフセット(VT_I4) 受信汎用 IO のサイズ(VT_I4)
位置と MiniIO + HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) HandIO 状態(VT_I4)
位置と MiniIO + HandIO 状態と受信汎用 IO パラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) 受信汎用 IO のオフセット(VT_I4)

		受信汎用 IO のサイズ（VT_I4） HandIO 状態（VT_I4）
位置と送信汎用 IO パラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 送信汎用 IO のオフセット（VT_I4） 送信汎用 IO のサイズ（VT_I4） 送信汎用 IO 状態（VT_UI1 | VT_ARRAY）
位置と送信汎用 IO パラメータと受信汎用 IO パラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 送信汎用 IO のオフセット（VT_I4） 送信汎用 IO のサイズ（VT_I4） 送信汎用 IO 状態（VT_UI1 | VT_ARRAY） 受信汎用 IO のオフセット(VT_I4) 受信汎用 IO のサイズ（VT_I4）
位置と送信汎用 IO パラメータ ＋ HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 送信汎用 IO のオフセット（VT_I4） 送信汎用 IO のサイズ（VT_I4） 送信汎用 IO 状態（VT_UI1 | VT_ARRAY） HandIO 状態(VT_I4)
位置と送信汎用 IO パラメータ ＋ HandIO 状態と受信汎用 IO パラメータ	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 送信汎用 IO のオフセット（VT_I4） 送信汎用 IO のサイズ（VT_I4） 送信汎用 IO 状態（VT_UI1 | VT_ARRAY） 受信汎用 IO のオフセット(VT_I4) 受信汎用 IO のサイズ（VT_I4） HandIO 状態(VT_I4)

表 4-3 slvMove コマンド戻り値フォーマット一覧

戻り値フォーマット	戻り値の型	戻り値
位置のみ(default)	VT_R8 | VT_ARRAY	位置(VT_R8 | VT_ARRAY)
位置と MiniIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4)

タイムスタンプと位置	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4)
タイムスタンプと位置と MiniIO 状態	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4)
位置と HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) HandIO 状態(VT_I4)
位置と MiniIO + HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) HandIO 状態(VT_I4)
タイムスタンプと位置と HandIO 状態	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) HandIO 状態(VT_I4)
タイムスタンプと位置と MiniIO + HandIO 状態	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) HandIO 状態(VT_I4)
位置と汎用 IO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY)
タイムスタンプと位置と汎用 IO 状態	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY)
位置と汎用 IO + HandIO 状態	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY) HandIO 状態(VT_I4)
タイムスタンプと位置と汎用 IO + HandIO 状態	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY) HandIO 状態(VT_I4)
位置と電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 電流値(VT_R8 | VT_ARRAY)
位置と MiniIO と電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4)

		電流値(VT_R8 | VT_ARRAY)
タイムスタンプと位置とMiniIOと電流値	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)
位置とHandIOと電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) HandIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)
位置とMiniIO+HandIOと電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) HandIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)
タイムスタンプと位置とHandIOと電流値	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) HandIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)
タイムスタンプと位置とMiniIO+HandIOと電流値	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) HandIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)
位置と汎用IOと電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY) 電流値(VT_R8 | VT_ARRAY)
タイムスタンプと位置と汎用IOと電流値	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4) 位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY) 電流値(VT_R8 | VT_ARRAY)
位置とMiniIO+汎用IOと電流値	VT_VARIANT | VT_ARRAY	位置(VT_R8 | VT_ARRAY) MiniIO 状態(VT_I4) 汎用 IO 状態(VT_UI1 | VT_ARRAY) 電流値(VT_R8 | VT_ARRAY)
タイムスタンプと位置	VT_VARIANT | VT_ARRAY	タイムスタンプ(VT_I4)

と汎用 IO+HandIO と電流値		位置(VT_R8 | VT_ARRAY) 汎用 IO 状態(VT_UI1 | VT_ARRAY) HandIO 状態(VT_I4) 電流値(VT_R8 | VT_ARRAY)

4.3. モードの説明

Slave Mode にはメッセージの処理仕様に応じて様々なモードが存在します. 表 4-4 にそれぞれのモードの概要を示します.

表 4-4 Slave Mode の概要

モード	パラメータ	バッファ数	バッファ空き待ち	備考
モード 0 同期・待機なし (RC7 該当無し)	0x0**	3 (バッファリング データは必ず 使用)	無	クライアントから送信されたメッセージをバッファにキューイングします. バッファの状態に応じたリターンコードを即時に返します.
モード 1 非同期 (RC7 の非同期に対応)	0x1**	1 (バッファリング時, データ上書き)	無	クライアントから送信されたメッセージでバッファを上書きし続けます.
モード 2 同期・待機あり (RC7 の同期に相当)	0x2**	3 (バッファリング データは必ず 使用) (RC7 はバッファ数は1)	有	クライアントから送信されたメッセージをバッファにキューイングします. バッファに空きができるまでリターンコードを返しません.

以降で, 各モードのメッセージ処理仕様を記します.

4.3.1. モード 0

モード 0 では, Robot_Execute "slvMove"によって送信された座標・姿勢データは, サーバのバッファにキューイングされます. サーバはキューイングされたバッファの状態に応じてクライアントにリターンコードを即時に返します.

バッファ状態	リターンコード
バッファに空きが 1 つ以上ある	S_OK(0x00000000)

バッファフル	S_ BUF_FULL (0x0F200501)
バッファオーバフロー	E_ BUF_FULL (0x83201483)

図 4-1 にモード 0 におけるクライアントとサーバの通信の流れを示します.

クライアントは一定周期ごとにメッセージ送信スレッドを発生させます. メッセージ送信スレッドでは, サーバからリターンコードとして S_BUF_FULL が返ってくるまで, サーバに"slvMove"メッセージを送信し続けます. S_BUF_FULL が返ってきた場合は, サーバがバッファフルの状態であるため, メッセージ送信スレッドを終了し, サーバがメッセージを処理するのを待ちます.

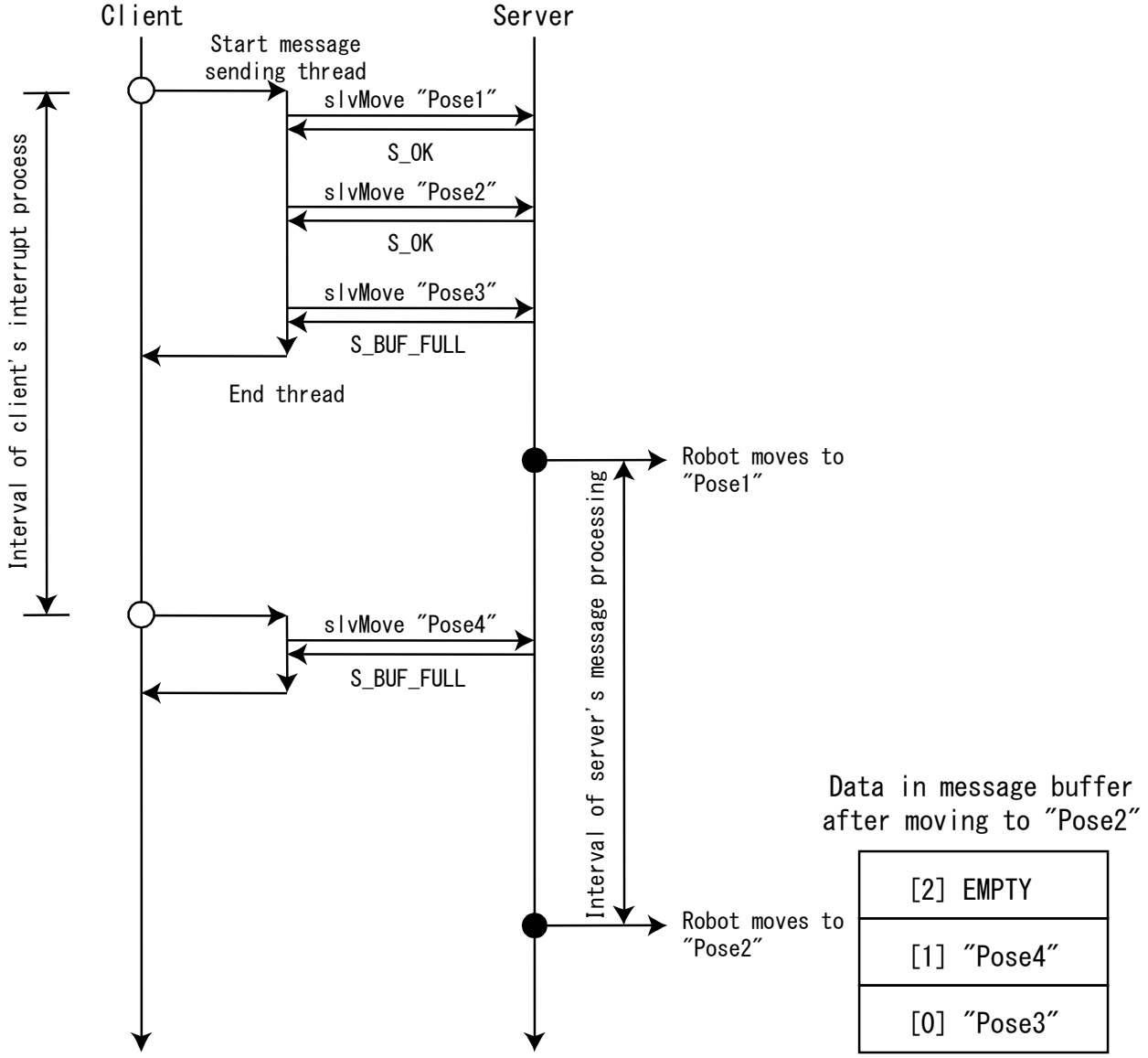


図 4-1 モード 0 の通信の流れ

もし，サーバからバッファオーバフローのメッセージが返ってきた場合は，送信した"slvMove"メッセージはバッファに積まれていません． 図 4-2 に示すように，サーバがメッセージを処理するのを待ってバッファオーバフローが返ってきた"slvMove"メッセージを再送する必要があります．

バッファオーバフローはエラーメッセージとして返ってきますが，コントローラ側ではエラーを発生しないため「4.7 エラー発生時の処理」に示す処理は必要ありません．

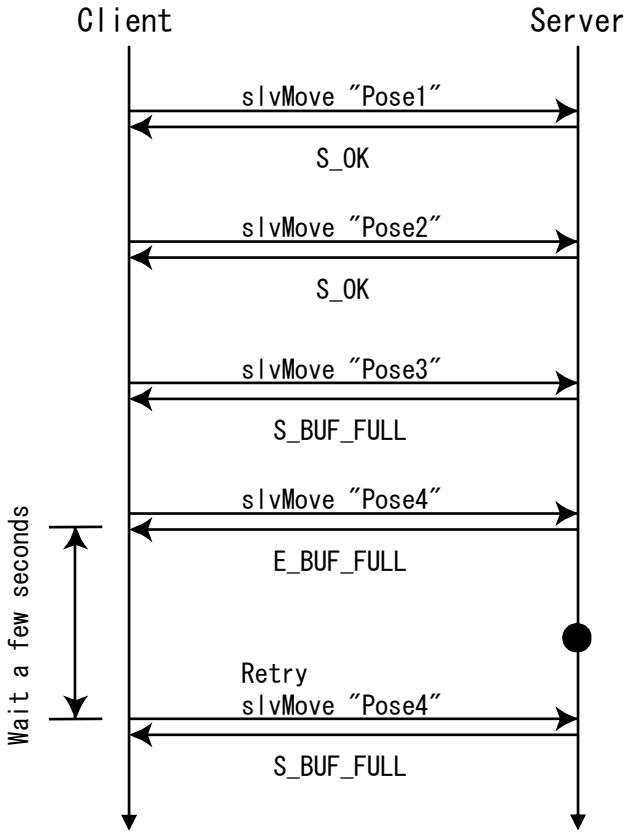


図 4-2 バッファオーバフロー発生時の処理

4.3.2. モード 1

モード 1 では，サーバのバッファサイズが1つに固定されます．Robot_Execute "slvMove"によって送信された座標・姿勢データは，サーバのバッファを上書きすることで格納されます．図 4-3 にモード 1 におけるクライアントとサーバの通信の流れを示します．

クライアントが送信した"slvMove"メッセージはバッファを上書きし続けるため，サーバが処理するメッセージはクライアントが直前に送信したメッセージとなります．

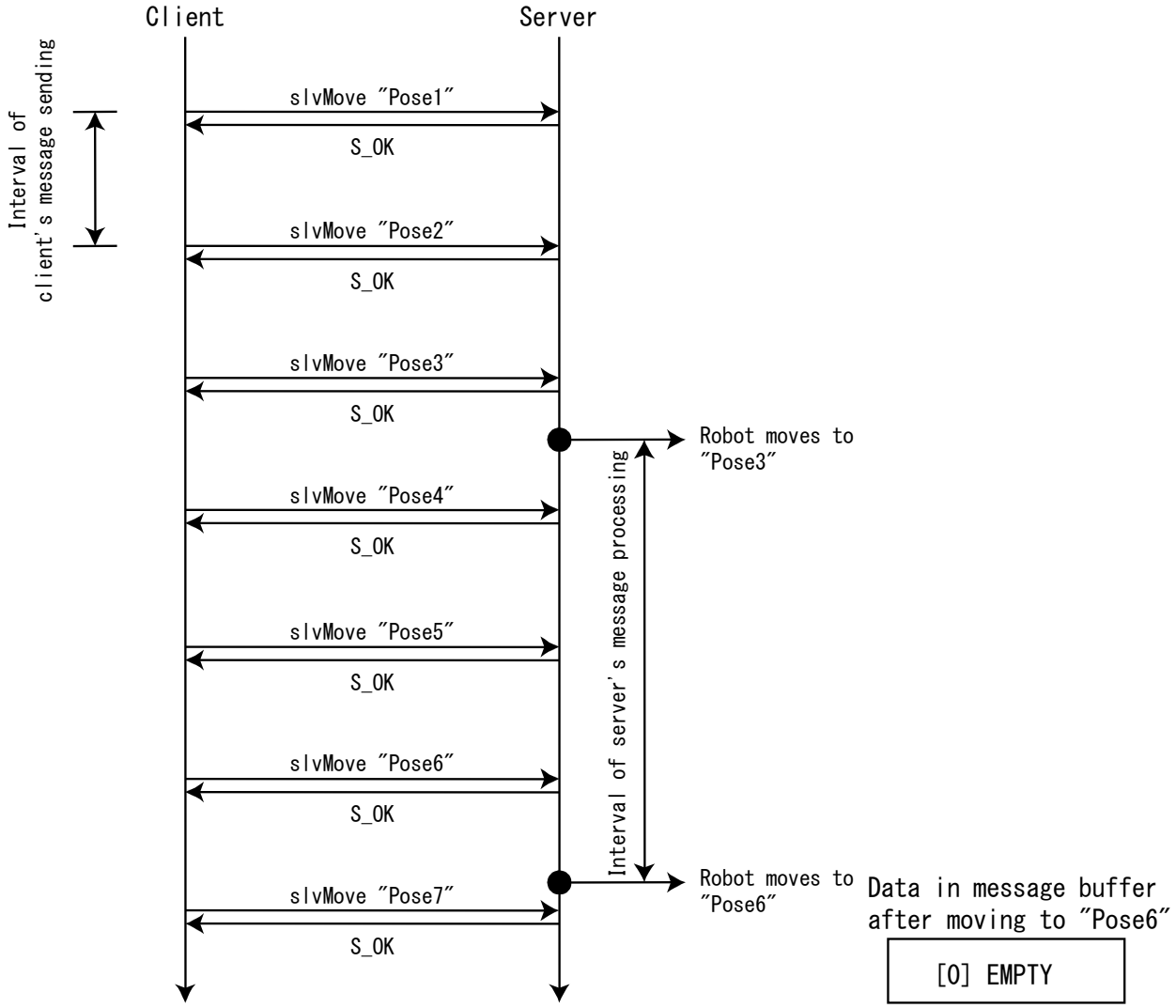


図 4-3 モード 1 の通信の流れ

4.3.3. モード 2

モード 2 では，モード 0 と同様に Robot_Execute "slvMove"によって送信された座標・姿勢データは，サーバのバッファにキューイングされます．サーバはキューイングされたバッファの状態に応じてクライアントにリターンコードを返します．モード 0 と異なる点は，バッファフルの状態で"slvMove"メッセージが送信された場合，サーバはバッファに空きができるまでリターンコードを返さないという点です．

図 4-4 にモード 2 におけるクライアントとサーバの通信の流れを示します．

バッファフルの状態で送信された"slvMove "Pose5""は，サーバが"Pose2"へロボットを移動させるまでリターンコードが返ってきません．そのため，クライアントはバッファに空きができるまで自動的に待機状態になります．これにより，クライアントはモード 0 であった「リターンコードを見てスレッドを終了させる」などの処理を実装することなく Slave Mode を実現することができます．

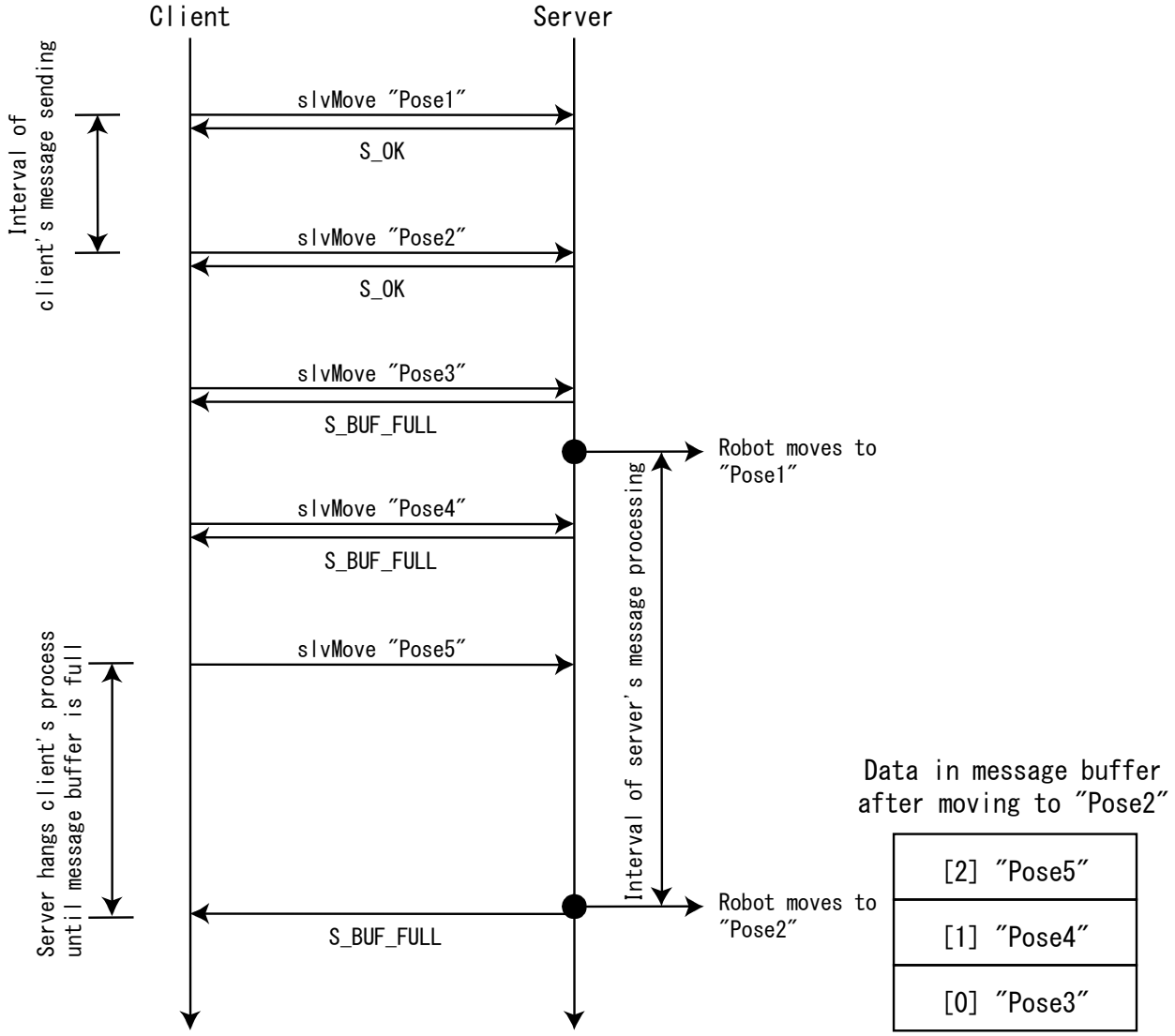


図 4-4 モード 2 の通信の流れ

4.4. 付加軸の扱い

Slave Mode で付加軸を制御するには slvMove コマンドパラメータの位置情報に付加軸オプションを追加します. 表 4-5 に付加軸オプションの詳細を示します.

表 4-5 付加軸オプション

位置形式	付加軸	位置情報
P 型	7 軸のみ	VT_R8 | VT_ARRAY (9 要素) (X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0x40, J7)
P 型	8 軸のみ	VT_R8 | VT_ARRAY (9 要素) (X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0x80, J8)

P 型	7 軸, 8 軸	VT_R8 | VT_ARRAY (10 要素) (X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0xC0, J7, J8)
J 型	7 軸のみ	VT_R8 | VT_ARRAY (9 要素) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0x40)
J 型	8 軸のみ	VT_R8 | VT_ARRAY (9 要素) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0x80)
J 型	7 軸, 8 軸	VT_R8 | VT_ARRAY (9 要素) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0xC0)
T 型	7 軸のみ	VT_R8 | VT_ARRAY (12 要素) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0x40, J7)
T 型	8 軸のみ	VT_R8 | VT_ARRAY (12 要素) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0x80, J8)
T 型	7 軸, 8 軸	VT_R8 | VT_ARRAY (13 要素) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0xC0, J7, J8)

4.5. バッファアンダーフローの扱い

Slave Mode は「4.3 モードの説明」に示したように，クライアントから送信された位置・姿勢データをバッファに保存し，一定周期でバッファの情報を読み出し動作生成を行います．バッファの情報を読み出す際にバッファが空（バッファアンダーフロー）である場合，実行中のモードによってサーバ側の挙動が異なります．表 4-6 に各モードにおけるバッファアンダーフロー時のサーバ側の挙動を示します．

表 4-6 各モードにおけるバッファアンダーフロー時のサーバ挙動

スレーブモード	ロボット動作状態	サーバ挙動	備考
モード 0	ロボット動作中	エラーを発生する 指令値生成遅延 (0x84201482)	スレーブモード解除
モード 0	ロボット停止中	エラーを発生しない	スレーブモード維持
モード 1	ロボット動作中	エラーを発生しない	現在位置停留命令を発行 スレーブモード維持
モード 1	ロボット停止中	エラーを発生しない	現在位置停留命令を発行 スレーブモード維持
モード 2	ロボット動作中	エラーを発生する 指令値生成遅延 (0x84201482)	スレーブモード解除
モード 2	ロボット停止中	エラーを発生しない	スレーブモード維持

ここでロボット停止中とは，ロボットの各軸現在速度がゼロであることを示し，そうでない状態をロボット動作中としています．

モード 0 およびモード 2 は，ロボット動作中にバッファが空になった場合，指令値生成が間に合わなかったと判断して，エラーとして「指令値生成遅延(0x84201482)」を発生します．ロボットを停止させる場合には，必ず同じ指令値を 2 回以上連続で送信して，指令速度をゼロにする必要があります．また，ロボットが十分に低速になっていない状態でこの操作を実行すると，急停止となりエラーとして「*軸指令加速度過大(0x8420404*)」が発生することがあります．

モード 1 はバッファが空であった場合，ロボット動作状態に関わらず現在位置に留まる命令を発行します．ロボットが十分に低速になっていない状態で現在位置に留まる命令が発行された場合，急停止となりエラーとして「*軸指令加速度過大(0x8420404*)」が発生することがあります．

4.6. Slave Mode の通信手順

図 4-5 に Slave Mode の通信手順を示します．Slave Mode にはコントローラオブジェクトとロボットオブジェクトが必要です．各オブジェクトのハンドラ取得までの手順・各オブジェクトを切断する以降の手順は「3.3 ロボット制御」を参照して下さい．以降で，各手順の詳細について説明します．

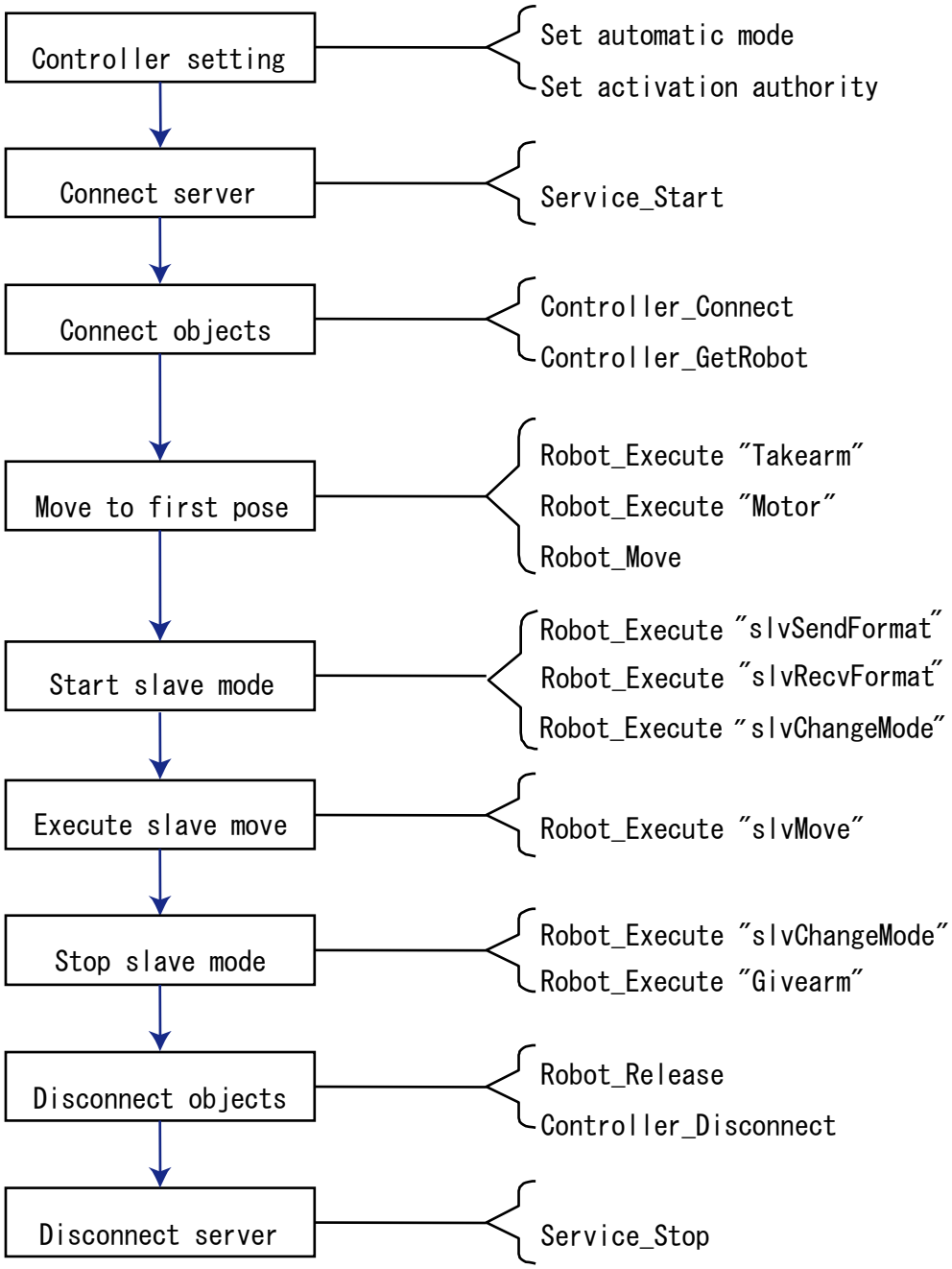


図 4-5 Slave Mode の流れ

4.6.1. 初期姿勢に移動

Slave Mode を開始する前に，ロボットを Slave Move で移動させる最初の座標・姿勢に移動させる必要があります．ロボットを移動させる手順の詳細は「3.3 ロボット制御」を参照して下さい．

Slave Mode を開始する前には，ロボットが停止状態である必要があります．また，ロボットが指定した初期姿勢へ完全に移動するために，移動命令 Robot_Move のオプションとして"@E"オプションを使用することをお勧めします．以下に"@E"オプションを使用してロボットを移動させるパケットの例を示します．

<table>
<tr><td colspan="4">Robot_Move 1, "@E P1", ""</td></tr>
<tr><td colspan="4">動作コマンド"MOVE 1, "@E P1" """を実行します.</td></tr>
<tr><td rowspan="11">送信
パケット</td><td colspan="3">クライアント→サーバ:
01 52 00 00 00 06 00 00 00 48 00 00 00 04 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00
00 03 00 01 00 00 00 01 00 00 00 14 00 00 00 08
00 01 00 00 00 0A 00 00 00 40 00 45 00 20 00 50
00 31 00 0A 00 00 00 08 00 01 00 00 00 00 00 00
00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>ロボットハンドル</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 03 00 00 00</td></tr>
<tr><td>lComp</td><td>補間指定</td><td>VT_I4</td><td>0x00000001</td></tr>
<tr><td></td><td colspan="3">0A 00 00
00 03 00 01 00 00 00 01 00 00 00</td></tr>
<tr><td>vntPose</td><td>ポーズ列</td><td>VT_BSTR</td><td>"@E P1"</td></tr>
<tr><td></td><td colspan="3">14 00 00 00 08
00 01 00 00 00 0A 00 00 00 40 00 45 00 20 00 50
00 31 00</td></tr>
<tr><td>bstrOption</td><td>動作オプション</td><td>VT_BSTR</td><td>""</td></tr>
<tr><td></td><td colspan="3">0A 00 00 00 08 00 01 00 00 00 00 00 00
00</td></tr>
<tr><td rowspan="5">受信
パケット</td><td colspan="3">サーバ→クライアント:
01 10 00 00 00 06 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="3">-</td></tr>
</table>

4.6.2. Slave Mode の開始・終了

Slave Mode の開始・終了を行うには, Robot_Execute "slvChangeMode"を用います. Slave Mode を開始する前にはクライアントがアーム制御権を取得している必要があります. アーム制御権の取得については「3.3 ロボット制御」を参照して下さい. Slave Mode 終了時には, サーバがメッセージバッファを全て処理してからモードを変更します. そのため, クライアントはメッセージ処理の間待ちが発生します. 以下に Slave Mode の開始および終了パケットの例を示します. ここでは, モード 0 で Slave Mode を開始します.

Robot_Execute "slvChangeMode", 0x001
Slave Mode をモード 0 で開始します.

送信 パケット	クライアント→サーバ: 01 54 00 00 00 08 00 00　00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 24 00 00 00 08 00 01 00 00 00 1A　00 00 00 73 00 6C 00 76 00 43 00 68 00 61 00 6E　00 67 00 65 00 4D 00 6F 00 64 00 65 00 0A 00 00　00 03 00 01 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00　00 03 00 00 00		
	bstrCommand	コマンド	VT_BSTR	"slvChangeMode"
		24 00 00 00 08 00 01 00 00 00 1A　00 00 00 73 00 6C 00 76 00 43 00 68 00 61 00 6E　00 67 00 65 00 4D 00 6F 00 64 00 65 00		
	vntParam	パラメータ	VT_I4	0x001
		0A 00 00　00 03 00 01 00 00 00 01 00 00 00		
受信 パケット	サーバ→クライアント: 01 1A 00 00 00 08 00 00　00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00　00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00　00		

Robot_Execute "slvChangeMode", 0x000				
Slave Mode を終了します.				
送信 パケット	クライアント→サーバ: 01 54 00 00 00 0A 00 00　00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00　00 03 00 00 00 24 00 00 00 08 00 01 00 00 00 1A　00 00 00 73 00 6C 00 76 00 43 00 68 00 61 00 6E　00 67 00 65 00 4D 00 6F 00 64 00 65 00 0A 00 00　00 03 00 01 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00　00 03 00 00 00		
	bstrCommand	コマンド	VT_BSTR	"slvChangeMode"
		24 00 00 00 08 00 01 00 00 00 1A　00 00 00 73 00 6C 00 76 00 43 00 68 00 61 00 6E　00 67 00 65 00 4D 00 6F 00 64 00 65 00		
	vntParam	パラメータ	VT_I4	0x000

		0A 00 00 00 03 00 01 00 00 00 00 00 00 00		
受信 パケット	サーバ→クライアント: 01 1A 00 00 00 0A 00 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

4.6.3. Slave Move

Slave Mode でロボットを移動するには, Robot_Execute "slvMove"を用います. 以下に Slave Mode パケットの例を示します. ここでは, P 型座標データを 1 パケット送信します. 実際に Slave Mode を用いたロボット動作を行う場合には,「4.3 モードの説明」に示す処理手順を行ってください.

Robot_Execute "slvMove"				
指定座標・姿勢データを送信します.				
送信 パケット	クライアント→サーバ: 01 7C 00 00 00 09 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 18 00 00 00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 00 4D 00 6F 00 76 00 65 00 3E 00 00 00 05 20 07 00 00 00 C3 F5 28 5C 8F C2 76 40 00 00 00 00 00 00 00 00 21 B0 72 68 91 68 71 40 00 00 00 00 00 80 66 40 80 8A 86 4A DC A5 0C 3D 00 00 00 00 00 80 66 40 00 00 00 00 00 00 14 40 04			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	コマンド	VT_BSTR	"slvMove"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 00 4D 00 6F 00 76 00 65 00		
	vntParam	パラメータ	VT_R8 | VT_ARRAY	364.16, 0, 278.5355, 180, 1.272222E-14, 180, 5
		3E 00 00 00 05 20 07 00 00 00 C3 F5 28 5C 8F C2 76 40 00 00 00 00 00 00 00 00 21 B0 72 68 91 68 71 40 00 00 00 00 00 80 66 40 80 8A 86 4A DC A5 0C 3D 00 00 00 00 00 80 66 40 00 00 00 00 00 00 14 40		

<table>
<tr><td>受信
パケット</td><td colspan="3">サーバ→クライアント:
01 5A 00 00 00 09 00 00 00 00 00 00 00 01 00 46
00 00 00 05 20 08 00 00 00 C1 0B B2 0D EB 4B D8
BC F0 D1 25 37 00 80 46 40 0D 97 E5 F5 FF 7F 56
40 26 BC 9E 96 1A 58 06 3D F6 FF 0E DD FF 7F 46
40 96 B0 06 42 EC 4B D8 BC 0F 00 00 00 89 11 40
00 FE FF FF FF 10 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>vntReturn</td><td>戻り値</td><td>VT_R8 | VT_ARRAY</td><td>-1.34873E-15, 45.0, 90, 9.922799E-15, 45, -1.348731E-15, 0, 0</td></tr>
<tr><td colspan="4">00 00 00 05 20 08 00 00 00 C1 0B B2 0D EB 4B D8 46
BC F0 D1 25 37 00 80 46 40 0D 97 E5 F5 FF 7F 56
40 26 BC 9E 96 1A 58 06 3D F6 FF 0E DD FF 7F 46
40 96 B0 06 42 EC 4B D8 BC 0F 00 00 00 89 11 40
00 FE FF FF FF 10 00 00 00</td></tr>
</table>

4.7. エラー発生時の処理

Slave Mode 中にエラーが発生した場合，Slave Mode は解除されティーチングペンダントに該当するエラーが表示されます．エラー発生後も Slave Mode を継続するためには，クライアントにエラー復帰処理を実装する必要があります．クライアントの復帰処理として必須な項目は①ティーチングペンダントのエラークリアと②Slave Mode の開始の2つです．

4.7.1. コントローラのエラークリア

コントローラのエラーが発生している状態では，Slave Mode を再開することができません．コントローラのエラーをクリアするには，ペンダントを操作してクリアする方法とb-CAP でエラークリアのパケットを送信する方法があります．b-CAP ではエラークリアは Controller_Execute（17）のコマンドとして実装されています．以下に，エラークリアパケットの例を示します．

<table>
<tr><td colspan="4">Controller_Execute "ClearError"</td></tr>
<tr><td colspan="4">ティーチングペンダントのエラーをクリアします．</td></tr>
<tr><td>送信
パケット</td><td colspan="3">クライアント→サーバ
01 4A 00 00 00 12 00 00 00 11 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 1E 00 00
00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65
00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00 06
00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>hController</td><td>コントローラハンドル</td><td>VT_I4</td><td>0x0000002</td></tr>
<tr><td colspan="4">00 00 00 03 00 01 00 00 00 02 00 00 00 0A</td></tr>
</table>

	bstrCommand	コマンド文字列	VT_BSTR	"ClearError"
		1E 00 00 00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00		
	vntParam	パラメータ	VT_EMPTY	
		06 00 00 00 00 00 01 00 00 00		
受信 パケット	サーバ→クライアント: 01 1A 00 00 00 12 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	vntReturn	戻り値	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

4.7.2. Slave Mode の開始

エラーが発生した時点で，Slave Mode が解除されています．そのため，Slave Mode を再度開始する必要があります．Slave Mode を開始する手順については「4.6Slave Mode の通信手順」を参照して下さい．

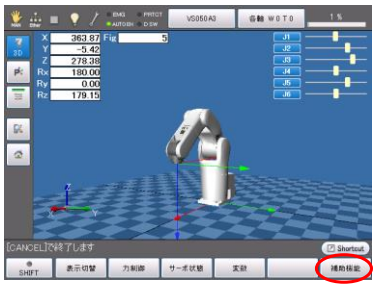
4.8. 指令速度/加速度の限界値に関する設定

Slave Mode では指令速度/加速度の限界値を設定することができます．限界値とは，Slave Move によって指定された姿勢を実現する際のロボットの速度/加速度の閾値であり，これを超えた場合「*軸送信指令速度過大(0x8420405*)」または「*軸指令加速度過大(0x8420404*)」が発生します．

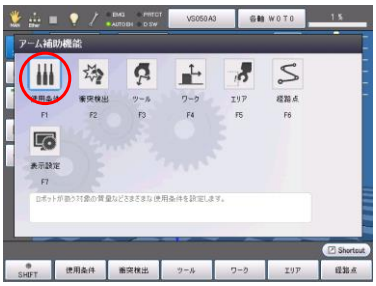
指令速度/加速度の限界値は，ティーチングペンダントの[アーム(F2)]メニュー ⇒ [補助機能(F6)] ⇒ [使用条件(F1)] ⇒ [153: b-CAP Slave 速度設定]で設定できます．

限界値に設定できる値は「0:サーボ限界」「1:サーボ限界(外部速度連動)」「2:指令限界」「3:指令限界(外部速度連動)」です．限界値の大小関係は(サーボ限界) > (指令限界)であり，外部速度連動を指定した場合は，各軸限界値に外部速度割合(加速度割合)を掛けた値が限界値となります．

[F2] ⇒



[F6] ⇒



[F1] ⇒

図 4-6 b-CAP Slave 速度設定

4.9. サンプルプログラム

以下に，ANSI-C 版サンプルライブラリを利用した Slave Mode のサンプルプログラムを示します．

サンプルプログラムは，モード 0 で Slave Mode を実行し，初期位置からサイン波 1 周期分ロボットを移動させます．初期位置として"J1"に格納されている座標を使用します．事前に"J1"にロボットの座標を指定して置いてください．また，IP は各コントローラで設定した値を用いてください．サンプルプログラムでは以下の設定値を用いて接続しています．

IP:192.168.0.1

List 4-1　bCapSlvMode.c

```
#if _MSC_VER > 1500
#include <stdint.h>
#else
#include "stdint.h"
#endif
#include <stdio.h>

#ifdef _USE_WIN_API
/*ifndef _USE_LINUX_API*/
#include <Windows.h>
#endif

#define _USE_MATH_DEFINES
#include <math.h>

#include "bcap_client.h"

#define PERIOD              (100)     /* Period Cycle */
#define AMPLITUDE   (10)      /* Amplitude */
#define TIMES               (3)

#define E_BUF_FULL  (0x83201483)

int main(void)
{
 int i, j, fd = 0;
```

```
long *plData;
uint32_t hCtrl = 0, hRob = 0;
double *pdData, dAng[8];
BSTR bstr1, bstr2, bstr3, bstr4;
VARIANT vntParam, vntRet;
HRESULT hr;

/* Open connection */
hr = bCap_Open_Client("tcp:192.168.0.1:5007", 1000, 3, &fd);
if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L",WDT=400");

/* Send SERVICE_START packet */
bCap_ServiceStart(fd, bstr1);

SysFreeString(bstr1);

bstr1 = SysAllocString(L"b-CAP");                                    // Name
bstr2 = SysAllocString(L"CaoProv.DENSO.VRC");  // Provider
bstr3 = SysAllocString(L"localhost");                    // Machine
bstr4 = SysAllocString(L"");                                                  // Option

/* Connect robot controller */
hr = bCap_ControllerConnect(fd, bstr1, bstr2, bstr3, bstr4, &hCtrl);

SysFreeString(bstr1);
SysFreeString(bstr2);
SysFreeString(bstr3);
SysFreeString(bstr4);

if (FAILED(hr)) goto END_PROC;

VariantInit(&vntParam);
bstr1 = SysAllocString(L"ClearError");

/* Clear Error */
hr = bCap_ControllerExecute(fd, hCtrl, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"Robot");     // Name
bstr2 = SysAllocString(L"");                    // Option

/* Get robot handle */
hr = bCap_ControllerGetRobot(fd, hCtrl, bstr1, bstr2, &hRob);

SysFreeString(bstr1);
SysFreeString(bstr2);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"Takearm");

VariantInit(&vntParam);
vntParam.vt = (VT_I4 | VT_ARRAY);
vntParam.parray = SafeArrayCreateVector(VT_I4, 0, 2);
SafeArrayAccessData(vntParam.parray, (void **)&plData);
plData[0] = 0; plData[1] = 1;
SafeArrayUnaccessData(vntParam.parray);

/* Get arm control authority */
```

```
hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

/* EXTSPEED 20% */
bstr1 = SysAllocString(L"ExtSpeed");

VariantInit(&vntParam);
vntParam.vt = (VT_R8 | VT_ARRAY);
vntParam.parray = SafeArrayCreateVector(VT_R8, 0, 1);
SafeArrayAccessData(vntParam.parray, (void **)&pdData);
pdData[0] = 20.0; // 20.0%
SafeArrayUnaccessData(vntParam.parray);

printf("External Speed = 20%¥n");
hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"Motor");

VariantInit(&vntParam);
vntParam.vt = VT_I4;
vntParam.lVal = 1;

/* Motor on */
bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

VariantInit(&vntParam);
vntParam.vt = VT_BSTR;
//vntParam.bstrVal = SysAllocString(L"@E J1");
vntParam.bstrVal = SysAllocString(L"@E J(45, 30, 120, 0, -60, 0)");

/* Move to first pose */
hr = bCap_RobotMove(fd, hRob, 1, vntParam, NULL);

VariantClear(&vntParam);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"CurJnt");

VariantInit(&vntParam);
vntParam.vt = VT_EMPTY;

/* Get current angle */
bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SafeArrayAccessData(vntRet.parray, (void **)&pdData);
memcpy(dAng, pdData, 8 * sizeof(double));
SafeArrayUnaccessData(vntRet.parray);

SysFreeString(bstr1);
VariantClear(&vntParam);
```

```
        VariantClear(&vntRet);

        bstr1 = SysAllocString(L"slvChangeMode");

        VariantInit(&vntParam);
        vntParam.vt = VT_I4;
        vntParam.lVal = 2;

        /* Start slave mode (Mode 0, J Type) */
        hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);
        VariantClear(&vntParam);
        VariantClear(&vntRet);

        if (FAILED(hr)) goto END_PROC;

        bstr1 = SysAllocString(L"slvMove");

        VariantInit(&vntParam);
        vntParam.vt = (VT_R8 | VT_ARRAY);
        vntParam.parray = SafeArrayCreateVector(VT_R8, 0, 8);

        for(j = 0; j < TIMES; j++) {
                 for(i = 0; i < PERIOD + 1; i++){
                          SafeArrayAccessData(vntParam.parray, (void **)&pdData);
                          memcpy(pdData, dAng, 8 * sizeof(double));
                          pdData[0] = dAng[0] - AMPLITUDE * cos(2 * M_PI*i / PERIOD) + AMPLITUDE;
                          pdData[1] = dAng[1] - AMPLITUDE * cos(2 * M_PI*i / PERIOD) + AMPLITUDE;
                          SafeArrayUnaccessData(vntParam.parray);

                          hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);
                          VariantClear(&vntRet);

                          /* if return code is not S_OK, then keep the message sending process waiting
for 8 msec */
                          if(hr != S_OK){
                                   Sleep(8);

                                   /* if return code is E_BUF_FULL, then retry previous packet */
                                   if(FAILED(hr)){
                                            if(hr == E_BUF_FULL){
                                                     i--;
                                            }else{
                                                     j = TIMES;
                                                     break;
                                            }
                                   }
                          }
                 }
        }

        /* Stop robot */
        bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);

        VariantClear(&vntParam);
        VariantClear(&vntRet);

        bstr1 = SysAllocString(L"slvChangeMode");

        VariantInit(&vntParam);
        vntParam.vt = VT_I4;
        vntParam.lVal = 0;
```

```
        /* Stop slave mode */
        bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);
        VariantClear(&vntParam);
        VariantClear(&vntRet);

        bstr1 = SysAllocString(L"Motor");

        VariantInit(&vntParam);
        vntParam.vt = VT_I4;
        vntParam.lVal = 0;

        /* Motor off */
        bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);
        VariantClear(&vntParam);
        VariantClear(&vntRet);

      END_PROC:
        if (hRob != 0) {
                bstr1 = SysAllocString(L"Givearm");

                VariantInit(&vntParam);
                vntParam.vt = VT_EMPTY;

                /* Release arm control authority */
                bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

                SysFreeString(bstr1);
                VariantClear(&vntParam);
                VariantClear(&vntRet);

                /* Release robot handle */
                bCap_RobotRelease(fd, &hRob);
        }
        if (hCtrl != 0) {
                /* Disconnect robot controller */
                bCap_ControllerDisconnect(fd, &hCtrl);
        }
        if (fd != 0) {
                /* Send SERVICE_STOP packet */
                bCap_ServiceStop(fd);

                /* Close connection */
                bCap_Close_Client(&fd);
        }

        return 0;
    };
```

4.10. 未対応機能

以下機能は b-CAP Slave では未対応です.

・複数台ロボットの動作

・バーチャルフェンス衝突防止

5. b-CAP Tester

ORiN2 SDK に付属しているツール b-CAP Tester を使用すると，コントローラと送受信されるパケットを確認することができます．

b-CAP Tester（b-CAPTester_RC8.exe）は以下のフォルダに格納されています．

ORiN2¥CAP¥b-CAP¥CapLib¥DENSO¥RC8¥Bin

図 5-1 に b-CAP Tester の機能紹介を示します．

コントローラ接続には表 5-1 に示すパラメータを設定します．

表 5-1 RC8 接続パラメータ

オプション	意味
Server=<IP アドレス>	接続するコントローラの IP アドレスを指定します．
Provider=<プロバイダ名>	RC8 に接続する場合は，"CaoProv.DENSO.VRC"を指定します．
Machine=<マシン名>	RC8 に接続する場合は，Server で指定した IP アドレスと同じ値を指定します．
Option[=<オプション文字列>]	リモートプロバイダに必要なオプション文字列を指定します．（デフォルト値：空文字列）
Message[=<True/False>]	メッセージ取得の有無． True：メッセージ取得あり（デフォルト） False：メッセージ取得なし
UDP[=<True/False>]	UDP による通信設定 True：UDP False：TCP（デフォルト） UDP 通信の場合パケットの最大サイズは 488 バイトになります．
Timeout=<タイムアウト時間>	送受信時のタイムアウト時間．（デフォルト：500 ms）
TORetry=<リトライ回数>	UDP 送受信時のリトライ回数．1～7（デフォルト：5） 1 以下の場合，1 として扱われます． 7 以上の場合，7 として扱われます． UDP のタイムアウト応答時間は，以下の式で算出されます． タイムアウト応答時間 = <Timeout>×<TORetry>
Debug[=<True/False>]	デバッグモードの指定 True：デバッグモード False：通常モード デバッグモードのときは，以下の変数を使用することができます．

	$LAST_SEND_PACKET$ $LAST_RECEIVE_PACKET$

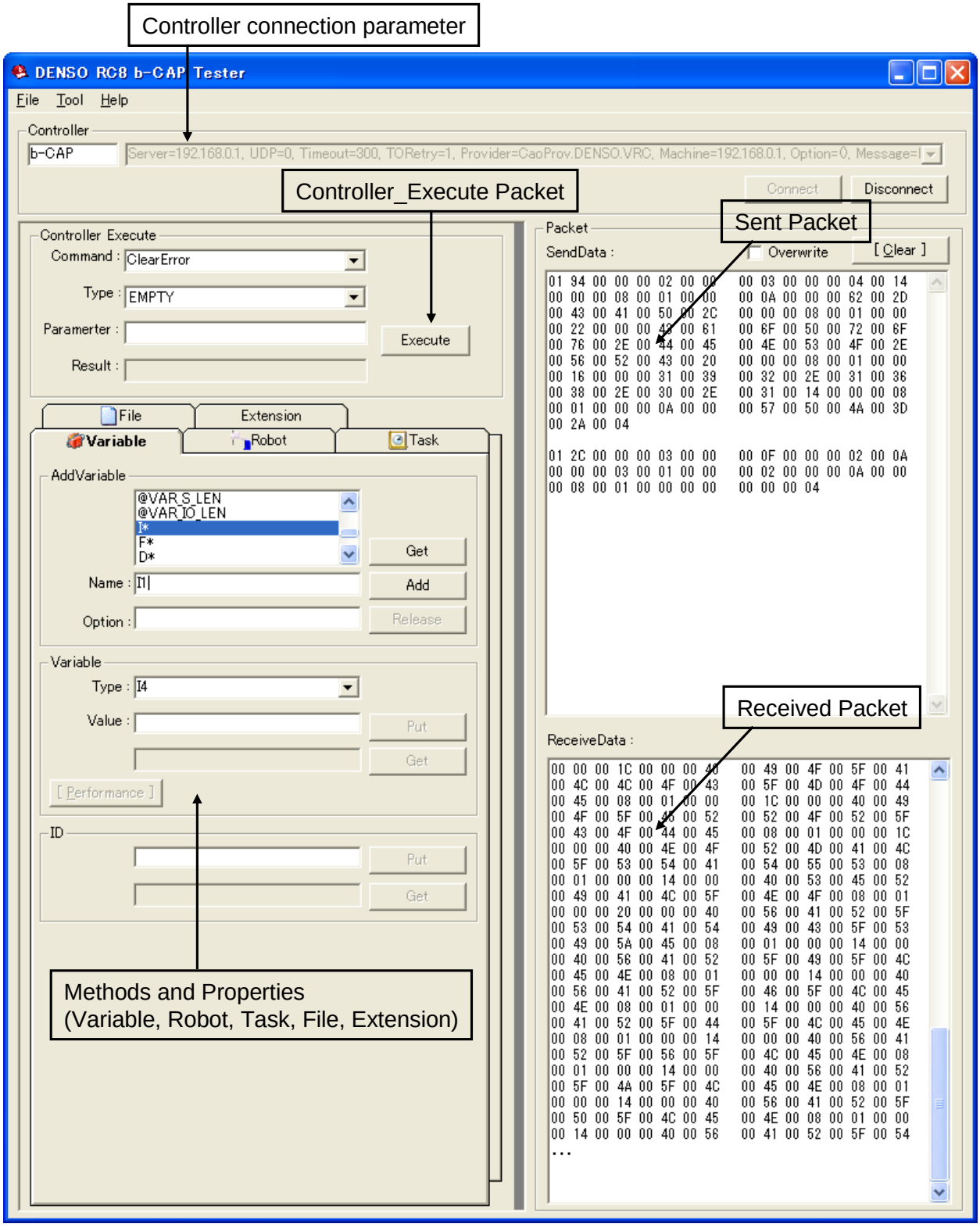


図 5-1 b-CAP Tester の機能紹介

5.1. b-CAP Tester での Slave Mode について

b-CAP Tester を用いて Slave Mode でロボットを動かすには以下の準備が必要です.

・制御ログを取得した WINCAPS3 プロジェクトファイル

b-CAP Tester の slvMove は制御ログの指令値を使用してロボットを動作させます.

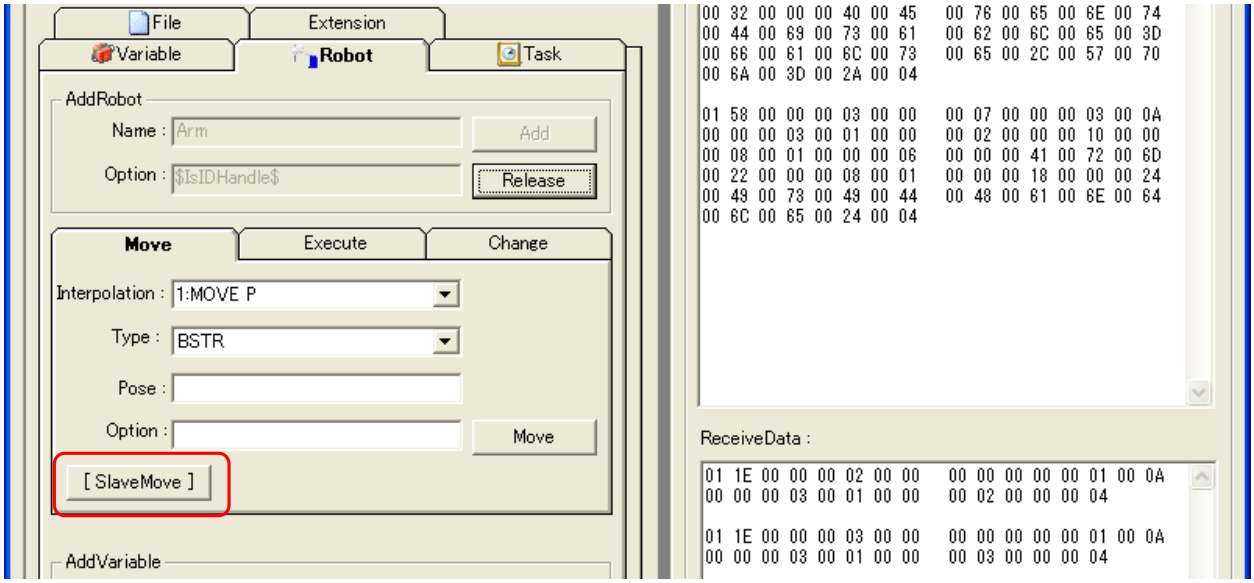
・コントローラの準備

コントローラを自動モードに, コントローラの起動権の設定をクライアント PC の IP に設定します.

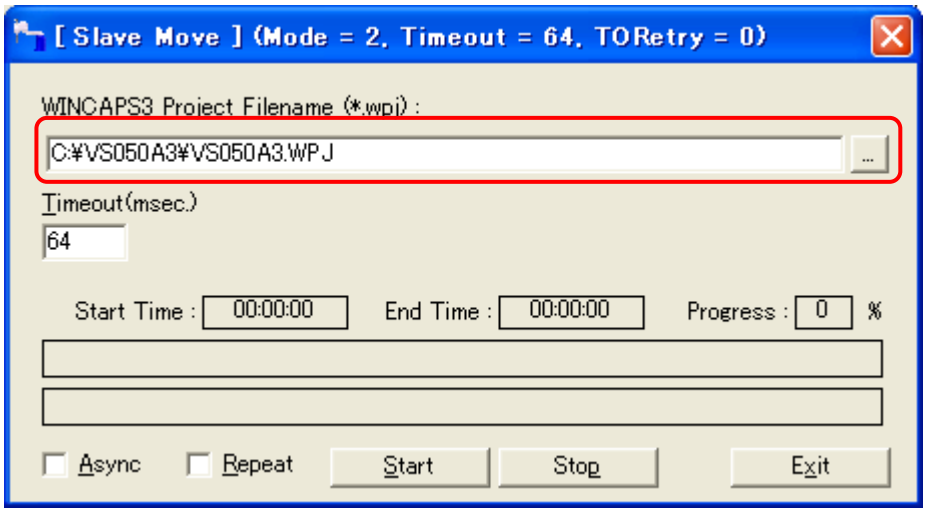
詳しくは「2.RC8 コントローラのセットアップ」を参照してください.

5.1.1. b-CAP Tester での Slave Mode の手順

1. ロボットオブジェクトへ接続した後, [Slave Move]ボタンを押し, Slave Move ウィンドウを立ち上げます.



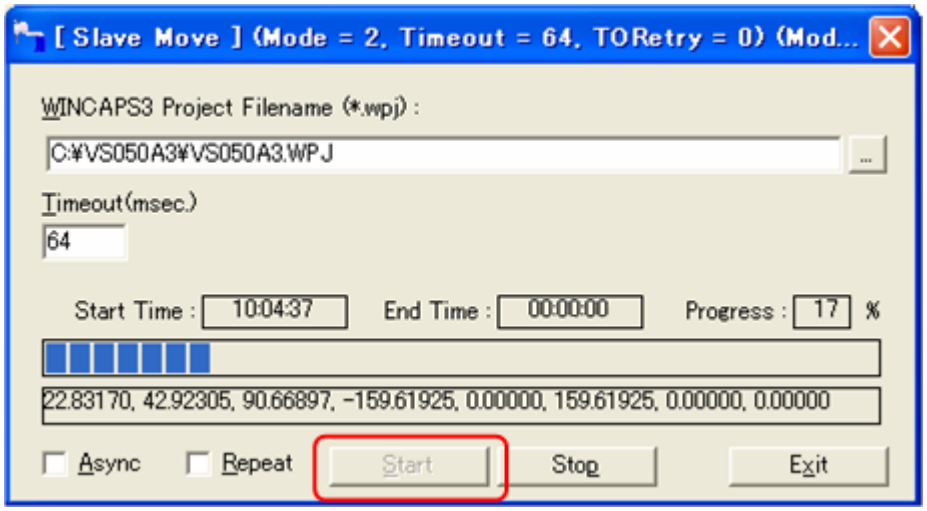
2. 制御ログを保存している WINCAPS3 プロジェクトを指定します.



3. Start ボタンを押すと動き始めます.
   ここでは, 以下の処理を自動で行っています.
   - 3.1 アーム制御権の取得
   - 3.2 初期姿勢に移動
   - 3.3 Slave Mode 開始
   - 3.4 Slave Move
   - 3.5 Slave Mode 終了
   - 3.6 アーム制御権の解放

モータ制御は自動では行いません. 事前に制御パケットを送信して Slave Mode を実行できる状態にしてください.

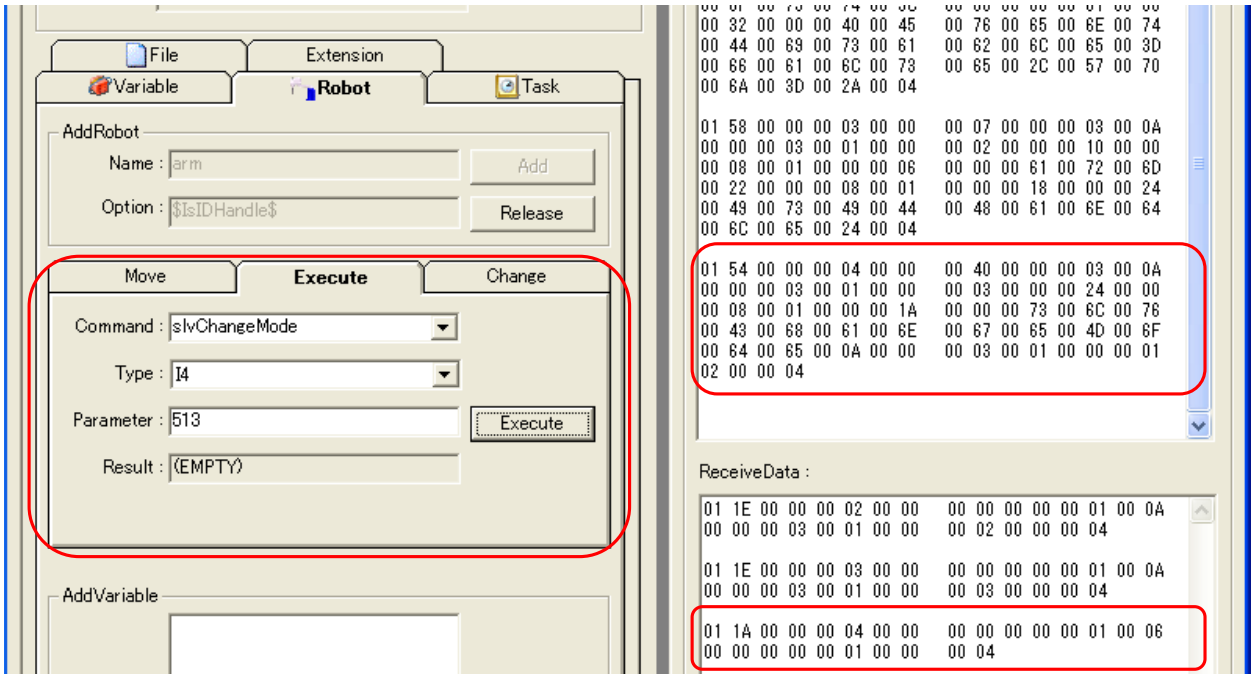
初期姿勢へ移動する際のタイムアウト時間はコントローラ接続時に設定した値に依存します. 初期姿勢移動でタイムアウトが発生する場合は, コントローラ接続時にタイムアウト時間を大きめに設定してください.



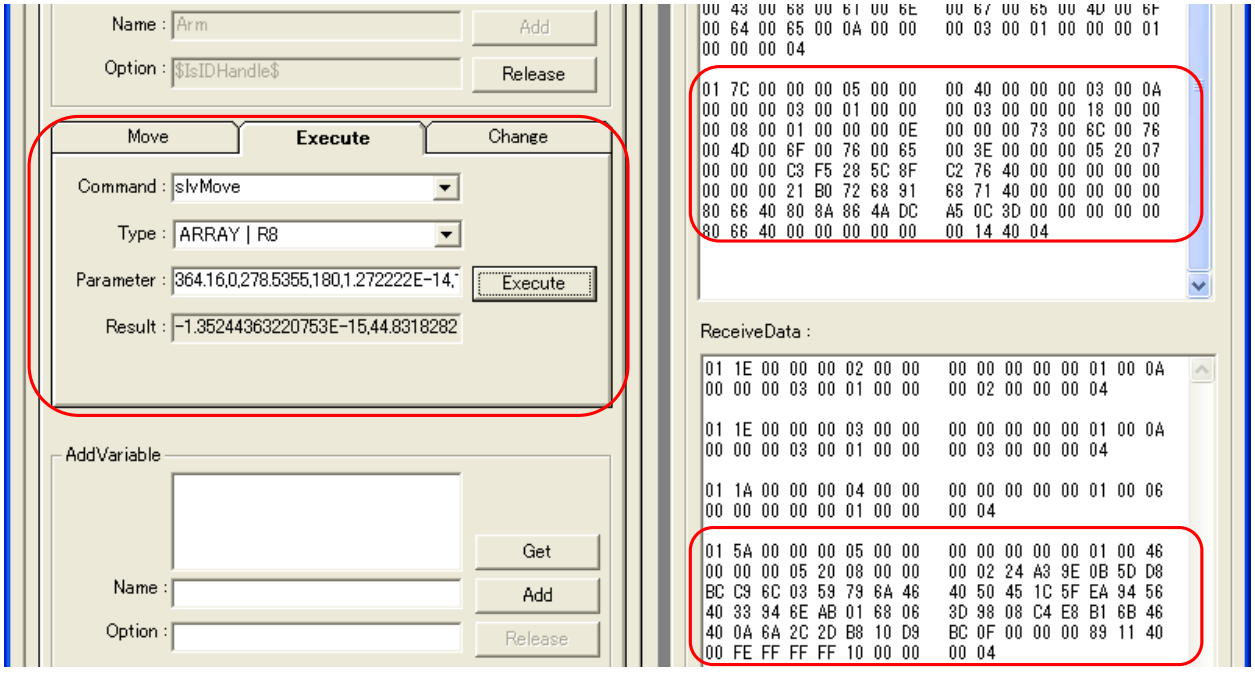
5.1.2. b-CAP Tester での Slave Mode のパケット確認

Slave Move ウィンドウを使用してロボットを動かす場合，送受信されるパケットは表示されません．パケットを確認する場合は，ロボットの Execute コマンドからコマンドを発行して確認することができます．

1. Execute タブの slvChangeMode でモード変更します．

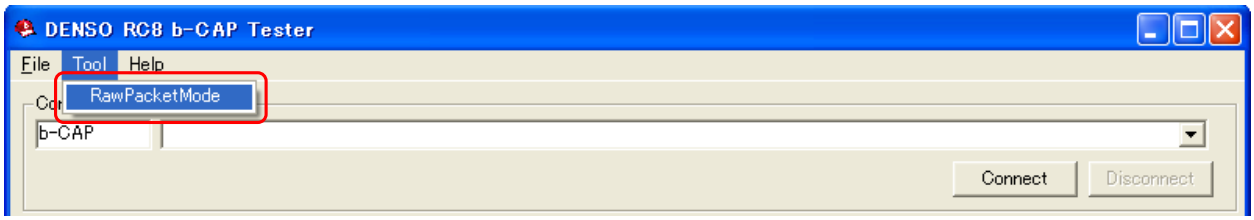


2. slvMove コマンドで動作させます．



5.2. Raw Packet Mode について

Raw Packet Mode では，手動で生成した b-CAP パケットを送信してコントローラを制御することができます．ここでは，Raw Packet Mode の使用方法を説明します．



5.2.1. コントローラ接続

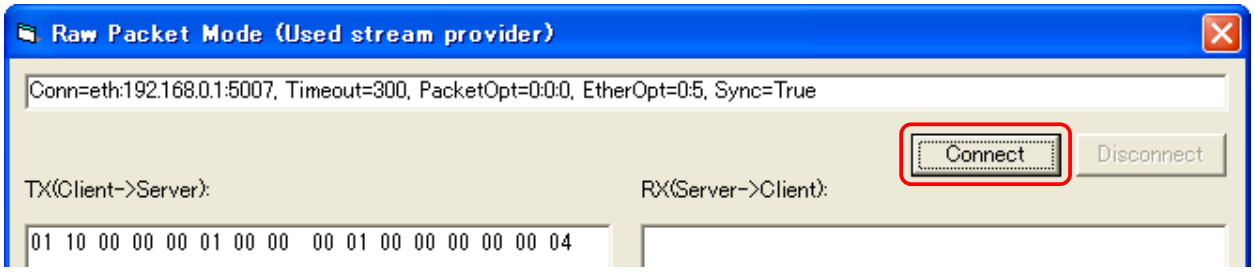
Raw Packet Mode でコントローラに接続するには，表 5-2 に示すパラメータを設定して[Connect]ボタンを押します．各パラメータの詳細については Stream プロバイダマニュアルを参照してください．

ORiN2¥CAO¥ProviderLib¥DENSO¥Stream¥Doc¥Stream_ProvGuide_ja.pdf

表 5-2 Raw Packet Mode 接続パラメータ

オプション	意味
Conn=eth:[<IP Address>[:<Port No>]]	接続するコントローラの IP アドレスを指定します．
Timeout=<タイムアウト時間>	送受信時のタイムアウト時間．（デフォルト：500 ms）
PacketOpt =[<Mode>[:<Header>[:<Term>]]]	<Mode>:通信データ変換． 1 ビット目：ISO 変換 2 ビット目：EIA 変換 3 ビット目：Unicode 変換 4 ビット目：テキストモード 5 ビット目：RoboTalk モード 6 ビット目：b-CAP モード <Header>:ヘッダ指定． ‘0’-なし，‘1’-ENQ(0x05) <Term>:ターミネータ指定． ‘0’-CR(0x0D)，‘1’-LF(0x0A)，‘2’-CR+LF(0x0D0A) b-CAP パケットを直接入力する場合は 0:0:0 を指定します．
EtherOpt =[<Mode>[:<ConnMax>]]	<Mode>:接続モード ‘0’-TCP クライアントモード，‘1’-TCP サーバモード， ‘2’-UDP クライアントモード，‘3’-UDP サーバモード <ConnMax>: TCP サーバモード時の最大クライアント数．

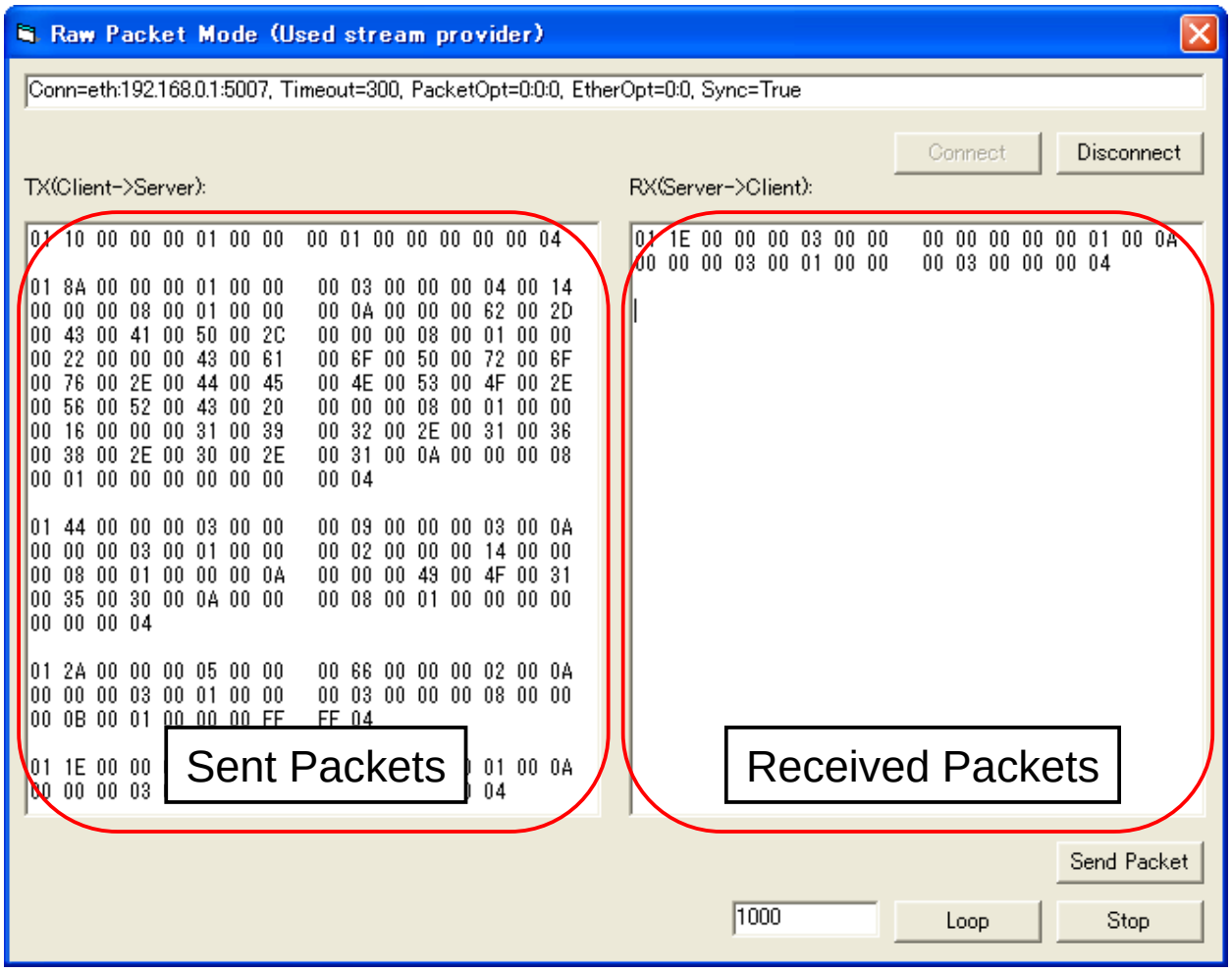
	Raw Packet Mode では 0:0 または 2:0 を指定します.
Sync=TRUE	同期モードの設定を行います. Raw Packet Mode では TRUE を指定します.



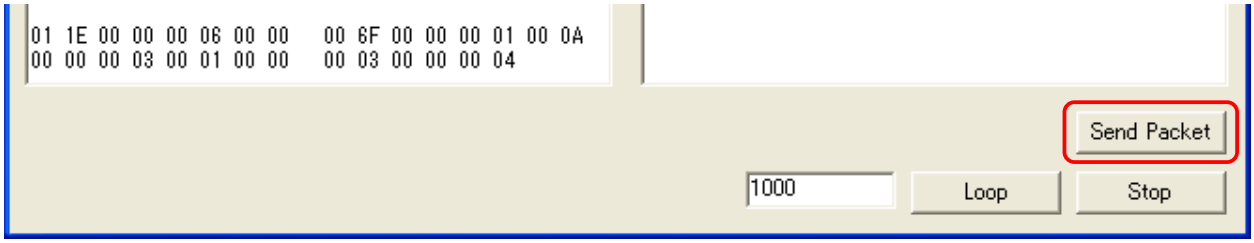
5.2.2. b-CAP パケットの送受信

Raw Packet Mode で b-CAP パケットを送信するには，左のテキストエリアに送信するパケットを書き込む必要があります．一度に複数のパケットを書き込み送信することもできます．

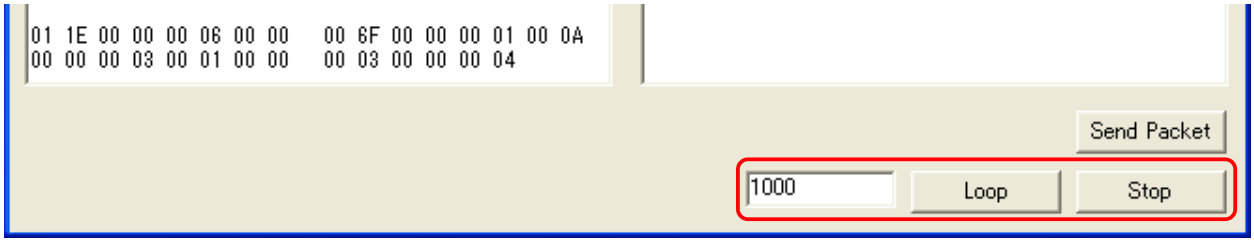
受信したパケットは右のテキストエリアに表示されます．複数のパケットを書き込み送信した場合，表示される受信パケットは最後に送信したパケットの応答結果となります．



送信するパケットの書き込み後，[Send Packet]ボタンを押すとパケットが送信されます．



書き込んだパケットを複数回送信したい場合は，送信回数を指定して[Loop]ボタンを押します．途中で送信を中止したい場合は[Stop]ボタンを押します．



5.3. b-CAP Tester での VT_ARRAY | VT_VARIANT 表記方法について

b-CAP Tester では VT_ARRAY|VT_VARIANT のパラメータを送る時に，以下の表現方法でパラメータを記述してください．

<データ型>, <データ列>

ここで，<データ型>には VARTYPE 型で表記される整数値を示します．表 5-3 に使用できるデータ型とその値を示します．

表 5-3 使用できるデータ型

データ型	値	意味
VT_I2	2	2 バイト整数型
VT_I4	3	4 バイト整数型
VT_R4	4	単精度浮動小数点型
VT_R8	5	倍精度浮動小数点型
VT_CY	6	通貨型
VT_DATE	7	日付型
VT_BSTR	8	文字列型
VT_BOOL	11	ブール型
VT_VARIANT	12	VARIANT 型

VT_UI1	17	バイト型
VT_ARRAY	8192	配列型

データ型が配列のときは, VT_ARRAY とデータ型の論理和で表記します.
データ列にはデータを文字列で表記します. 配列データの表記は", "(カンマ)で区切って表記します.

図 5-2 に Robot_Move の Pose での例を示します. Robot_Move では Pose として VT_ARRAY | VT_VARIANT を指定することができます. 第一配列に VT_R8 | VT_ARRAY(8197)で座標・姿勢データ(8197, 0, 0, 0, 0, 0, 0, 5)を, 第二配列に VT_I4(3)で変数型(0)を, 第三配列に VT_I4(3)でパス(-2)を記述しています.

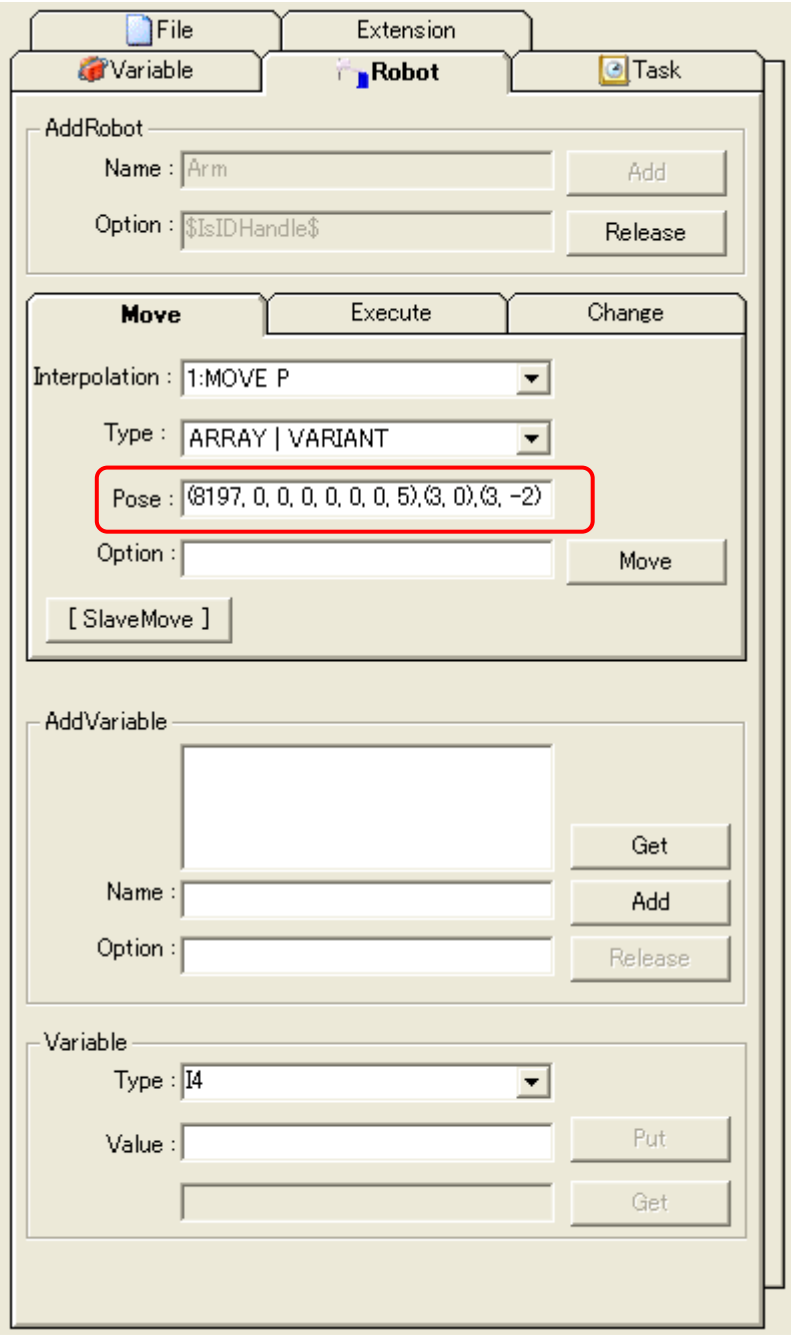


図 5-2　Robot_Move の Pose 表記例

付録A. b-CAP 関数 ID と CAO インターフェースの対応表

関数 ID	関数名	CAO インターフェース名	説明
1	Service_Start	CCaoWorkspace::AddController	サーバサービスの開始
2	Service_Stop	CCaoWorkspaces::Remove	サーバサービスの停止
3	Controller_Connect		コントローラとの接続
4	Controller_Disconnect		コントローラとの切断
5	Controller_GetExtension	CCaoController::AddExtension	コントローラの拡張ボード取得
6	Controller_GetFile	CCaoController::AddFile	コントローラのファイル取得
7	Controller_GetRobot	CCaoController::AddRobot	コントローラのロボット取得
8	Controller_GetTask	CCaoController::AddTask	コントローラのタスク取得
9	Controller_GetVariable	CCaoController::AddVariable	コントローラの変数取得
10	Controller_GetCommand	CCaoController::AddCommand	コントローラのコマンド取得
11	Controller_GetExtensionNames	CCaoController::get_ExtensionNames	コントローラの拡張ボード名一覧取得
12	Controller_GetFileNames	CCaoController::get_FileNames	コントローラのファイル名一覧取得
13	Controller_GetRobotNames	CCaoController::get_RobotNames	コントローラのロボット名一覧取得
14	Controller_GetTaskNames	CCaoController::get_TaskNames	コントローラのタスク名一覧取得
15	Controller_GetVariableNames	CCaoController::get_VariableNames	コントローラの変数名一覧取得
16	Controller_GetCommandNames	CCaoController::get_CommandNames	コントローラのコマ

			ンド名一覧取得
17	Controller_Execute	CCaoController::Execute	コントローラの拡張関数の実行
18	Controller_GetMessage	CCaoController::AddMessage	コントローラのイベントメッセージ取得
19	Controller_GetAttribute	CCaoController::get_Attribute	コントローラの属性値取得
20	Controller_GetHelp	CCaoController::get_Help	コントローラのヘルプ文字列取得
21	Controller_GetName	CCaoController::get_Name	コントローラの名前取得
22	Controller_GetTag	CCaoController::get_Tag	コントローラのタグ情報取得
23	Controller_PutTag	CCaoController::put_Tag	コントローラのタグ情報設定
24	Controller_GetID	CCaoController::get_ID	コントローラの ID 取得
25	Controller_PutID	CCaoController::put_ID	コントローラの ID 設定
26	Extension_GetVariable	CCaoExtension::AddVariable	拡張ボードの変数取得
27	Extension_GetVariableNames	CCaoExtension::get_VariableNames	拡張ボードの変数名一覧取得
28	Extension_Execute	CCaoExtension::Execute	拡張ボードの拡張関数実行
29	Extension_GetAttribute	CCaoExtension::get_Attribute	拡張ボードの属性値取得
30	Extension_GetHelp	CCaoExtension::get_Help	拡張ボードのヘルプ文字列取得
31	Extension_GetName	CCaoExtension::get_Name	拡張ボードの名前取得
32	Extension_GetTag	CCaoExtension::get_Tag	拡張ボードのタグ情報取得
33	Extension_PutTag	CCaoExtension::put_Tag	拡張ボードのタグ情報設定
34	Extension_GetID	CCaoExtension::get_ID	拡張ボードの ID 取

			得
35	Extension_PutID	CCaoExtension::put_ID	拡張ボードの ID 設定
36	Extension_Release	CCaoExtension::Release	拡張ボードの解放
37	File_GetFile	CCaoFile::AddFile	ファイルの別ファイル取得
38	File_GetVariable	CCaoFile::AddVariable	ファイルの変数取得
39	File_GetFileNames	CCaoFile::get_FileNames	ファイルの別ファイル名一覧取得
40	File_GetVariableNames	CCaoFile::get_VariableNames	ファイルの変数名一覧取得
41	File_Execute	CCaoFile::Execute	ファイルの拡張関数実行
42	File_Copy	CCaoFile::Copy	ファイルのコピー
43	File_Delete	CCaoFile::Delete	ファイルの削除
44	File_Move	CCaoFile::Move	ファイルの移動
45	File_Run	CCaoFile::Run	ファイルの実行
46	File_GetDateCreated	CCaoFile::get_DateCreated	ファイルの作成日時取得
47	File_GetDateLastAccessed	CCaoFile::get_DateLastAccessed	ファイルの最終アクセス日時取得
48	File_GetDateLastModified	CCaoFile::get_DateLastModified	ファイルの最終更新日時取得
49	File_GetPath	CCaoFile::get_Path	ファイルのパス取得
50	File_GetSize	CCaoFile::get_Size	ファイルのサイズ取得
51	File_GetType	CCaoFile::get_Type	ファイルのファイルタイプ取得
52	File_GetValue	CCaoFile::get_Value	ファイルの内容取得
53	File_PutValue	CCaoFile::put_Value	ファイルの内容設定
54	File_GetAttribute	CCaoFile::get_Attribute	ファイルの属性取得

55	File_GetHelp	CCaoFile::get_Help	ファイルのヘルプ文字列取得
56	File_GetName	CCaoFile::get_Name	ファイルの名前取得
57	File_GetTag	CCaoFile::get_Tag	ファイルのタグ情報取得
58	File_PutTag	CCaoFile::put_Tag	ファイルのタグ情報設定
59	File_GetID	CCaoFile::get_ID	ファイルの ID 取得
60	File_PutID	CCaoFile::put_ID	ファイルの ID 設定
61	File_Release	CCaoFile::Release	ファイルの解放
62	Robot_GetVariable	CCaoRobot::AddVariable	ロボットの変数取得
63	Robot_GetVariableNames	CCaoRobot::get_VariableNames	ロボットの変数名一覧取得
64	Robot_Execute	CCaoRobot::Execute	ロボットの拡張関数実行
65	Robot_Accelerate	CCaoRobot::Accelerate	ロボットの ACCEL 文実行
66	Robot_Change	CCaoRobot::Change	ロボットの CHANGE 文実行
67	Robot_Chuck	CCaoRobot::Chuck	ロボットの GRASP 文実行
68	Robot_Drive	CCaoRobot::Drive	ロボットの DRIVE 文実行
69	Robot_GoHome	CCaoRobot::GoHome	ロボットの GOHOME 文実行
70	Robot_Halt	CCaoRobot::Halt	ロボットの HALT 文実行
71	Robot_Hold	CCaoRobot::Hold	ロボットの HOLD 文実行
72	Robot_Move	CCaoRobot::Move	ロボットの MOVE 文実行
73	Robot_Rotate	CCaoRobot::Rotate	ロボットの ROTATE 文実行
74	Robot_Speed	CCaoRobot::Speed	ロボットの SPEED/JSPEED 文

			実行
75	Robot_Unchuck	CCaoRobot::Unchuck	ロボットの REELASE 文実行
76	Robot_Unhold	CCaoRobot::Unhold	ロボットの HOLD 文解除
77	Robot_GetAttribute	CCaoRobot::get_Attribute	ロボットの属性値取得
78	Robot_GetHelp	CCaoRobot::get_Help	ロボットのヘルプ文字列取得
79	Robot_GetName	CCaoRobot::get_Name	ロボットの名前取得
80	Robot_GetTag	CCaoRobot::get_Tag	ロボットのタグ情報取得
81	Robot_PutTag	CCaoRobot::put_Tag	ロボットのタグ情報設定
82	Robot_GetID	CCaoRobot::get_ID	ロボットの ID 取得
83	Robot_PutID	CCaoRobot::put_ID	ロボットの ID 設定
84	Robot_Release	CCaoRobot::Release	ロボットの解放
85	Task_GetVariable	CCaoTask::AddVariable	タスクの変数取得
86	Task_GetVariableNames	CCaoTask::get_VariableNames	タスクの変数名一覧取得
87	Task_Execute	CCaoTask::Execute	タスクの拡張関数実行
88	Task_Start	CCaoTask::Start	タスクの開始
89	Task_Stop	CCaoTask::Stop	タスクの停止
90	Task_Delete	CCaoTask::Delete	タスクの削除
91	Task_GetFileName	CCaoTask::get_FileName	タスクの元ファイル名
92	Task_GetAttribute	CCaoTask::get_Attribute	タスクの属性取得
93	Task_GetHelp	CCaoTask::get_Help	タスクのヘルプ文字列取得
94	Task_GetName	CCaoTask::get_Name	タスクの名前取得
95	Task_GetTag	CCaoTask::get_Tag	タスクのタグ情報取得
96	Task_PutTag	CCaoTask::put_Tag	タスクのタグ情報設定

97	Task_GetID	CCaoTask::get_ID	タスクの ID 取得
98	Task_PutID	CCaoTask::put_ID	タスクの ID 設定
99	Task_Release	CCaoTask::Release	タスクの解放
100	Variable_GetDateTime	CCaoVariable::get_DateTime	変数のタイムスタンプ取得
101	Variable_GetValue	CCaoVariable::get_Value	変数の値取得
102	Variable_PutValue	CCaoVariable::put_Value	変数の値設定
103	Variable_GetAttribute	CCaoVariable::get_Attribute	変数の属性値取得
104	Variable_GetHelp	CCaoVariable::get_Help	変数のヘルプ文字列取得
105	Variable_GetName	CCaoVariable::get_Name	変数の名前取得
106	Variable_GetTag	CCaoVariable::get_Tag	変数のタグ情報取得
107	Variable_PutTag	CCaoVariable::put_Tag	変数のタグ情報設定
108	Variable_GetID	CCaoVariable::get_ID	変数の ID 取得
109	Variable_PutID	CCaoVariable::put_ID	変数の ID 設定
110	Variable_GetMicrosecond	CCaoVariable::get_Microsecond	変数のタイムスタンプ(ミリ秒)取得
111	Variable_Release	CCaoVariable::Release	変数の解放
112	Command_Execute	CCaoCommand::Execute	コマンドの実行
113	Command_Cancel	CCaoCommand::Cancel	コマンドのキャンセル
114	Command_GetTimeout	CCaoCommand::get_Timeout	コマンドのタイムアウト時間取得
115	Command_PutTimeout	CCaoCommand::put_Timeout	コマンドのタイムアウト時間設定
116	Command_GetState	CCaoCommand::get_State	コマンドの状態取得
117	Command_GetParameters	CCaoCommand::get_Parameters	コマンドのパラメータ取得
118	Command_PutParameters	CCaoCommand::put_Parameters	コマンドのパラメータ設定
119	Command_GetResult	CCaoCommand::get_Result	コマンドの実行結果取得
120	Command_GetAttribute	CCaoCommand::get_Attribute	コマンドの属性値取

			得
121	Command_GetHelp	CCaoCommand::get_Help	コマンドのヘルプ文字列取得
122	Command_GetName	CCaoCommand::get_Name	コマンドの名前取得
123	Command_GetTag	CCaoCommand::get_Tag	コマンドのタグ情報取得
124	Command_PutTag	CCaoCommand::put_Tag	コマンドのタグ情報設定
125	Command_GetID	CCaoCommand::get_ID	コマンドの ID 取得
126	Command_PutID	CCaoCommand::put_ID	コマンドの ID 設定
127	Command_Release	CCaoCommand::Release	コマンドの解放
128	Message_Reply	CCaoMessage::Reply	イベントメッセージの応答
129	Message_Clear	CCaoMessage::Clear	イベントメッセージのクリア
130	Message_GetDateTime	CCaoMessage::get_DateTime	イベントメッセージのタイムスタンプ取得
131	Message_GetDescription	CCaoMessage::get_Description	イベントメッセージの説明文取得
132	Message_GetDestination	CCaoMessage::get_Destination	イベントメッセージの送信先取得
133	Message_GetNumber	CCaoMessage::get_Number	イベントメッセージのメッセージ番号取得
134	Message_GetSerialNumber	CCaoMessage::get_SerialNumber	イベントメッセージのシリアル番号取得
135	Message_GetSource	CCaoMessage::get_Source	イベントメッセージの送信元取得
136	Message_GetValue	CCaoMessage::get_Value	イベントメッセージの値取得
137	Message_Release	CCaoMessage::Release	イベントメッセージの解放