四足歩行ロボット TITANⅧ（広大改）
&
三本指三関節ロボット Hiroshima Hand 用

ロボット言語

T.I.P.

(*Tiny Interpreter-type Programming language for general robots*)

仕様書

Ver.1.0　：1999.6.1～　　白井達也
Ver.1.1　：1999.6.14～　白井達也
Ver.2.0　：2000.2.24～　白井達也
Ver.2.1　：2000.4.25～　白井達也
Ver.3.0　：2000.5.26～　白井達也
Ver.3.1　：2000.8.15～　白井達也

目次

1　仕様の変更履歴および予定

1-1 これまでの履歴

Ver.1.0:　1999 年 6 月 1 日～
Ver.1.1:　1999 年 6 月 1 4 日～

- ＥＭＧ＿ＯＮ／ＯＦＦ
- ＡＢＳ，ＩＮＴ，ＳＱＲＴなど数値演算追加
- ＧＯＳＵＢ／ＲＥＴＵＲＮ／ＩＦ＿ＧＯＳＵＢ追加
- A_MAX,V_MAX，OMEGA_MAX，OMEGA_DOT_MAX
  および各 SLOW,FAST,NORMAL を追加.
- T_MAX，FIRMLY/SOFT/NORMAL を追加.
- VEL_CONTROL，OMEGA_CONTROL を追加.
- OUTPUT VARIABLES を追加.
- プログラム構成の二行目に”プログラム説明”を追加.
- イベントフラグ（HALT_REQUEST）を追加
- REL_PTP,REL_PTP_ANGLE を追加
- A_MAX_RATE/V_MAX_RATE/OMEGA_MAX_RATE/OMEGA_DOT_MAX_RATE/T_MAX_RAT
  E コマンドを追加
- EXIT コマンド追加
- ラベルのサンプルを修正（最後に：を付け忘れ）

Ver.2.0:　2000 年 2 月 24 日～

- 足裏センサ、触角センサ(PAW1,2,3,4,ANTENNA)の入力に対応
  （AD_IN, LOG）
- イベントフラグに KB_HIT を追加
- FLUSH_BUF コマンドを追加
- パラメータ調節系コマンドの頭に，SET_を追加.
- SET_JUST_POSITION/SET_NEAR_POSITION コマンドを追加
- 算術演算命令の追加（SIN, COS, TAN, ATAN, ATAN2, ASIN, ACOS, DEG2RAD, RAD2DEG, NORM, UNIT_VECT,）.
- イベントフラグに，INPOS/NEAR を追加
- SLEEP コマンドとの追加
- イベントフラグの論理反転(!)の追加
- STAND_UP/SIT_DOWN コマンドの内容変更（速度を SET_V_MAX_SLOW とする）
- 代入演算(+=, -=, *=, /=）を全ての算術演算で使用可能とする.
- 符号反転コマンド（Ａ　＝　－　Ｂ）を削除.（Ａ　＝　－Ｂで代用可能なため）
- MESSAGE コマンドに，汎用変数(VARnn)表示機能を追加

Ver.2.1:　2000 年 4 月 25 日～

- HiroshimaHand への対応開始
- SPECIAL コマンドの追加
- SPECIAL_FLAGn の追加
- WAIT_INPOS/NEAR の仕様を変更
- DETECT_FLOOR を追加（HiroshimaHand 専用）
- LAMP_ON/LAMP_OFF コマンドの追加
- SILENT_ON/OFF コマンドの追加
- COMPLIANCE コマンドの仕様変更
- COMPLIANCE → COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE に変更
- SET_ANGLE_STIFFNESS コマンドを追加
- サンプリング関連の設定変更コマンドを追加（SET_SAMPLING_TIME/FREQ）
- DETECT_OBJECT を追加（HiroshimaHand 専用）
- 実装された SPECIAL コマンドに関する説明を追加
- FORCE コマンドを追加(HiroshimaHand)
- COMPLIANCE_POSITION / REL_COMPLIANCE_POSITION コマンドを追加
- SET_POSITION_STIFFNESS コマンドを追加
- TEACH_POS コマンドの名称を TEACH_POSITION コマンドに変更
  (互換性を残すために，TEACH_POS も使用可能)
- TEACH_POSITION コマンドで指定可能なターゲットを脚先座標のみから，関節の座標まで拡張.
- TEACH_FORCE コマンドを追加(HiroshimaHand のみ)
- イベントフラグ INPOS/NEAR の仕様を変更
  （関節位置の位置制御への対応＋INPOS_J?／NEAR_J?を追加）
- TitanVIII も DETECT_FLOOR に対応させる.
- LOG_ONE_SHOT コマンドの追加

Ver.3.0:　2000 年 7 月 26 日～

- 変数の型を VAR, POS, AGL, TRQ の４つから VAR, POS, JOINT の３つに変更する.
- 自由な変数名を使用可能にする（ただし全ての変数を宣言：DEF_***，する必要がある）
- マルチステートメントに対応
- 制御構文の追加（IF-THEN-ELSE-ENDIF, WHILE-WEND, DO-WHILE, FOR-NEXT）
- 文字コードに対応する定数（CHAR_***）の追加
- SUB_ROUTINE-SUBEND / CALL を追加
- REM をコマンドとして解釈するように変更.
- MESSAGE コマンドの汎用変数置換機能の書式を変更（自由な変数名に対応したため，従来の”番号”で指定する方法が使えなくなったため）.
- 警告(WARNING.LOG)を出力するようにした．具体的には，宣言したが使われていない変数が存在する，値を代入したが参照されない変数が存在した，場合に発生する.
- NORM 演算コマンドの引数を座標型変数（ＰＯＳ）に加えて，関節型も可とした．例えば，指（脚）の関節の変化量を正の値で得てることで，指（脚）が静止しているか，動いているかを判定するなどの用途に利用できる.
- IF-THEN-ELSE-ENDIF に，ELSEIF を追加．格段にプログラム記述の自由度が向上するのだが，何で入れ忘れたのだろう.

Ver.3.1:　2000 年 8 月 15 日～

- SET_INPOS/SET_NEAR を SET_JUST_POSITION, SET_NEAR_POSITION_POSITION に変更.
- SET_JUST_ANGLE, SET_NEAR_POSITION_ANGLE を追加
- WAIT_NEAR で引数を取るのを止める（以前は SET_NEAR_POSITION コマンドが無かったため）.
- PTP/REL_PTP/COMPLIANCE_POSITION/REL_COMPLIANCE_POSITION の tip_if.cpp 内の処理を共通の関数にまとめる.
- PTP_ANGLE/REL_PTP_ANGLE/COMPLIANCE_ANGLE/REL_COMPLIANCE_ANGLE の tip_if.cpp 内の処理を共通の関数にまとめる.
- PTP/REL_PTP/COMPLIANCE_POSITION/REL_COMPLIANCE_POSITION, PTP_ANGLE/REL_PTP_ANGLE/COMPLIANCE_ANGLE/REL_COMPLIANCE_ANGLE に，高速移動フラグ（ＨＳ）を追加．加速度制限（Ａ__ＭＡＸ）を用いず高速に移動することを指示する．実装方法は各ロボットに任される.
- エラーチェックの強化：数式評価でのチェック不足（○：Ａ＝Ｂ＋Ｃ　×：Ａ＝Ｂ＋Ｃ＋Ｄ）
- エラーチェックの強化：CALL コマンドのラベルのチェックミス（CALL ラベル：　セミコロンを付けずに，この後にコマンドをマルチステートメントのつもりで記述してしまうと，エラーにならずに読み飛ばされていた）
- HiroshimaHand も TARGET_ANTENNA に対応（パームの ON/OFF 触覚センサ，ON=１，OFF=0）．イベントフラグに ANTENNA_ON を追加.
- HiroshimaHand 用に TARGET_FLOOR を追加．取り合えず PTP,REL_PTP で使用可能とする．スライダーの位置を X,Y,Z 座標で指示可能とする．HiroshimaHand 用に，SET_V_MAX, SET_A_MAX で TARGET_FLOOR を使用可能とする.
- TEACH_ANGLE, TEACH_TORQUE でターゲットが脚（指）単位だけだったのを関節単位で指定できるように拡張する.
- SET_NEAR_ANGLE/SET_JUST_ANGLE で，ターゲットを ALL とした場合に，引数一つ（SET_NEAR_ANGLE ALL 3.0）の形式で全関節を設定できるように拡張した.
- SET_COMPLIANCE_TORQUE_LIMIT, SET_COMPLIANCE_FORCE_LIMIT を追加．コンプライアンス制御では目標角度，位置との差に比例したトルク，力が発生するが，差が大きすぎると過大なトルク，力が発生してしまうので，これを防ぐのが目的.
- ROT_X, ROT_Y, ROT_Z コマンドの追加.
- FOR-NEXT コマンドの使用を変更．(FOR A B TO C -> FOR A = B TO C STEP D)
  "="を入れることで可読性を向上し，STEP（刻み）を追加することで機能向上（省略化）

1-2 次バージョンで取り入れる予定の機能

1-2-1 エラー処理の徹底

現状では，エラー処理に手を抜いている個所が非常に多い（なんでもかんでも，format_error にしていたりする個所がある）．エラーの種別に合わせたエラーを発生するように徹底する（準備はされているが，使用していないだけである）．

1-2-2 DIRECT / TRAJECTORY コマンド（仮称）の追加

PTP や COMPLIANCE_POSITION, PTP_ANGLE や COMPLIANCE_ANGLE などは，V_MAX, A_MAX, OMEGA_MAX, OMEGA_DOT_MAX の制限下にあるため，小刻みに目標位置（角度）を変更しても即座には反応してくれない．TIP で軌道を計画してもそれに追従しない．そこで，DIRECT コマンド後は，位置（角度）に関して軌道計画（速度／角速度の台形制御）を行わずに，一気にＰ制御で目標位置（角度）へ直行するように制御モードを切り替える．関節単位，脚（指）単位で指定できるように，コマンドの与え方を工夫する．

とりあえず，ＨＳを PTP コマンドなどの引数に追加することで対処(TitanVIII は未対応)．

1-2-3 TIP_SPEED_UP コマンドの追加

1Cycle で実行可能な TIP コマンド数（現在は 1 コマンド）を増やすことで TIP の処理速度を向上させる．数値計算のみならば負荷は少ないため，サンプリング速度に影響しない（キー入力や MESSAGE 出力などは時間が掛かる）．

1-2-4 F_MESSAGE コマンド

MESSAGE コマンドと同じ書式で特定のファイル(FOUT.LOG)に出力．ログと異なり自由な文字列を出力できる半面，直にファイル出力を行うため，サンプリングが乱れる欠点がある．要は使い分け．文字列の長さの制限は MESSAGE コマンドよりも長くする予定．タブ出力(¥t)や改行(¥n)にも対応しなくてはならない．

1-2-5 MESSAGE/F_MESSAGE コマンドの変数置換機能の拡張

汎用変数以外の変数も出力できるように拡張する予定．POS 変数を構造体で指定した場合は"X Y Z"，関節型変数を構造体で指定した場合は"J1 J2 J3 ..."のように，半角スペース 1 文字で区切って出力する．

1-2-6 SET_ANGLE_STIFFNESS / SET_POSITION_STIFFNESS の拡張

現在，SET_ANGLE_STIFFNESS は関節毎にパラメータを設定しなくてはならないのを指（脚）単位で指定できるように拡張する．SET_POSITION_STIFFNESS は，３つの引数で X,Y,Z 方向のバネを設定するが，これも座標型変数一つで指定できるように拡張する．

1-2-7 数式の解析時のエラー処理を改良する.

定義されていない変数を使用した数式（例：VAR10 = VAR1 + 1）に対して，”VAR10 は Unknown command”であると解釈してしまう．せめて，＝（または+=,-=, /=, *=）を見て，第一要素が変数であると判断するようにする．ついでに言うと，Unknown command のエラーには，エラーとなったコマンド（文字列）が何か？程度はメッセージに含ませるべきである．

1-2-8 LOCK/UNLOCK コマンド

特定の関節／脚（指）などに対する制御モードおよび目標値の変更を禁じる．

1-2-9 マルチスレッド化

少なくとも２つ程度のスレッドが走るようになると良い．スレッド間の通信には変数を用いれば良い．

1-2-10 定数の宣言（DEFINE）

書き込み不可の変数にするか，TIP.CPP での解析時に置換するか?

1-2-11 特殊変数の強化（TEACH 系コマンドの特殊変数化）

特殊変数(@AGL_Jnn, @AGL_Fn, @POS_Fn, @TRQ_Jnn, @TRQ_Fn, @FORCE_Fn)を参照すれば，常に現在の角度，位置，トルク，力が得られるならば，TEACH_????は不要になる．もちろん，代入は不可.

@TIMERn も追加.

1-2-12 ソフトウェアリミット

関節角度にソフトウェア上でリミットを設定する．制御目標角度だけではなく，現在の角度がソフトウェアリミットの範囲内にある場合はどうするか，などの問題を考えると，結構，仕様を厳密に作らなくてはならない.

1-2-13 タイマーコマンド

START_TIMER, STOP_TIMER, RESTART_TIMER, RESET_TIMER

1-2-14 ネイティブコードとの通信用変数領域 (VAR_SPECIAL_n, POS_SPECIAL_n, JNT_SPECIAL_n)

SPECIAL コマンドで呼び出されるネイティブコードとＴＩＰプログラムとの間で相互にデータをやり取りする手段がない．SPECIAL_FLAGn（ON/OFF のみ，かつネイティブ→ＴＩＰの一方向のみ）だけでは不便.

２　設計思想

本仕様書は以下に掲げる設計思想に基づき策定された.

① 誰でも簡単にTitanⅧの動作を安全かつ簡単に設計できること.
② 高度過ぎる機能は導入しないことで，利用者の学習を容易とする.
③ 汎用的なｎ関節ｍ本足（指）ロボットの自動プログラム用インタプリタとして流用できるように最大限の配慮を行う．具体的には足本数を４本と限定したコーディングをする代わりに定数(#define)を用いたり，Titan のみに固有と思われる命令は特化コマンド(Special Command)として分類する.

３ プログラム構造

プログラムは以下に示す形式を必ず守ること．

１行目：バージョン番号
２行目：プログラム説明
３行目以降：コマンド　引数１　引数２　．．．　引数ｎ
最終行：ＥＮＤ

① テキスト形式
② １行には，”；”（セミコロン）で区切られたマルチステートメント[1]の記述を可能とする．
③ 本体プログラムと異なるバージョン番号のプログラムは，読み込み時にエラーとする．
④ プログラム中にＥＮＤコマンドがあったならば，それ以降のプログラムは無視される．ＥＮＤ行の存在しないプログラムは異常とみなし，エラーとする．
⑤ １行の長さは半角で１０２３文字（バイト）以内とする．
⑥ 半角空白およびタブ記号(¥t)を区切り子として解釈する．
⑦ コマンドは基本的に，“コマンド　引数１　引数２．．．”の形式である．
（注意）コマンドの種類によって引数の個数／有無は異なる．また，変数の演算および代入は第１引数の部分が演算子になる．各引数は，区切り子（空白，タブ，コンマ記号[2]）で区切られる．
⑧ 行中に//記号があった場合，それ以降はコメントとみなされ，無視される．";"で区切ってマルチステートメントとしても全て無視される．（例：　//これはコメント　；KEY_IN）
⑨ 同じくコメントを記述するための REM コマンドがあるが，REM コマンドはコマンドとして解釈される．実行時には 1cycle 消費するので，NOP(Not a Operation)として用いることができる．また，FORMAL.TMP にも"REM"として記述されるので，FORMAL.TMP の可読性が少々向上する（ただし，REM の内容は FORMAL.TMP には記述されない）．
⑩ 警告ログ(WARNINIG.LOG)には，例えば定義されたが使用されていない変数名や値が代入されるが参照されない変数名などが出力される．実行時ではなく，中間言語への変換時にのみ出力される．実行時の異常は全て非常停止となる．

以下は注意事項である．

① 全角空白は半角空白２個に変換された後に解析される．
② アルファベットの小文字は大文字に変換された後に解析される（MESSAGE コマンドの（文字列）を除く）．
③ コメント行および MESSAGE コマンドの（文字列）を除き，基本的に全角および 2byte 半角文字の使用を禁じる．
④ 空行は無視される．
⑤ 引数に定数（数値）が与えられた場合，たとえ整数形式で表記されていたとしても，内部では，一旦，double 型に変換した上で格納される．

[1] Ver.3 以前はシングルステートメント．Ver.3 以降は，さらにラベル行にも命令文を記述可能．
[2] Ver.3 以降

4 プログラミング言語モードの仕組み

4-1 Auto プログラムと TIP プログラム

Titan プログラム内(strategy.c)に記述する形式(ネイティブコード)の自動運転プログラムを Auto プログラム，テキストファイル(TITAN.PRG)で記述されたスクリプト(TIP)に従って動作する形式の自動運転プログラムを TIP プログラムと呼ぶ．それぞれ一長一短があるため，状況によって使い分ける．

Auto プログラム

（長所）
- 三角関数やヤコビ行列の演算など，高度な演算が可能である．
- 処理速度が速い．

（短所）
- Ｃ言語によって記述する必要があるため，Ｃ言語を習得している必要があり，Ｃ言語特有のトラブルに見舞われる可能性がある．
- Titan プログラム全体に対する理解が必要である．
- 動作を変更するたびに再コンパイルの必要がある．
- 最大で９種類の動作プログラムしか１つの実行形式には組み込めない．
  （９動作もあれば十分との判断もある）

ただし，動作プログラムは strategy.h および strategy.c を差し替えて再コンパイルすることで，９個以上のプログラムを容易に切り替えることができる．

TIP プログラム

（長所）
- TIP の仕様を理解すれば，誰でも簡単に Titan の動作プログラムが記述できる．
- 内容の異なるプログラムファイル(TITAN.PRG)を読み込ませるだけで違う動作に切り替えることが可能である．
- status の管理（strategy.h 内）が不要で読み易いコードで記述できる．
- 未定義変数，未代入変数の参照（＝エラー）を発見できる．

（短所）
- あまり高度なコマンドは装備していないので，複雑な条件分岐やハードに密着した処理は記述できない．
- 処理速度が Auto に比べると遅い．理由は「１サイクルに１コマンドしか実行しない」ためである．例えば値を代入（VAR01=VAR02+VAR03）するだけでも１サイクル（たとえば 20msec）必要である[3]．

[3] ただしこの制約は，仕様を変えて，１サイクルにｎコマンドを実行する，などの設定を設ければ緩和できる．あるいは各コマンド毎に”重み”（≒サイクル数）を決めて，それが一定数量に達するまで一度に実行してしまう，などの手もあるが，ちょっと凝り過ぎだろう．

4-2　動作の仕組み

ほぼ完全に Auto プログラムと同じである．
・原点復帰完了フラグＯＮ（＝初期姿勢）の状態で自動モードを選択
・Program No.として０（ゼロ）を選択する．

(1)TIP プログラムの読み込みを開始する．ファイルが存在しない場合はエラー(File not found)とする．
(2)バージョン番号を読み込み，本体プログラム(titan.exe)のバージョンと比較し，異なる場合はエラー(Wrong version number)とする．
(3)Path1: 中間言語ファイル（FORMAL.TMP)の生成
　A. それ以外の行に関しては，END 行まで以下の処理を繰り返し，中間言語ファイルに書き出して行く．
　　① TIP プログラム(TITAN.PRG)から１行読み込む．もし１行の文字数が１０２３文字よりも多い場合はエラー(Format error)とする．
　　①. 文字列中の全角スペースを半角スペース×２に変換する．
　　②. 行頭,行末の全角／半角空白およびタブ(¥t)を削除する．
　　③. 行中の//より以降の文字列を削除する．
　　④. ＲＥM行および空行は読み飛ばす．
　　⑤. MESSAGE コマンド行を除き,余分な区切り子を削除し，タブ（¥t）に置き換える．ラベル行のラベル名とコロン（”:”）の間の区切り子を取り除く．
　　⑥. MESSAGE コマンド行は何も手を加えない（Path2 で処理する）．
　　⑦. 編集し終えた文字列を中間言語ファイルに書き出す．
　B. TIP プログラム，中間言語ファイル共に閉じる．
(4)Path2: ラベルテーブルの作成およびプログラム行数のカウント，文字列テーブルの必要数をカウント
　A. プログラムステップ番号を０にリセットする．
　B. ラベルテーブルファイル(LABEL.TMP)を（上書き/テキスト形式で）開く．
　C. 中間言語ファイル(FORMAL.TMP)を（読み込み/テキスト形式で）開く．
　D. max_label_number, string.max_num をゼロクリアし，以下，END 行まで繰り返す．
　　① 中間言語ファイル(FORMAL.TMP)から１行読み込む．
　　② ラベル行ならば，ラベル名とプログラムステップ番号をラベルテーブルファイル(LABEL.TMP)に書き出す．max_label_number を＋１する．
　　③ 文字列関連のコマンドの場合(MESSAGE コマンド)は，string.max_num を＋１する．
　　④ プログラムステップ番号を＋１する．
　E. 中間言語ファイル，ラベルテーブルファイル共に閉じる．
　F. この時のプログラムステップ番号をプログラム行数(max_step_no)として記憶する．
　G. max_label_number,に基づき,必要な数のラベル変換テーブルをメモリ上に（一時的に）確保する．
　H. ラベルテーブルファイル(LABEL.TMP)の中身をラベル変換テーブルに転記する．
　I. ラベル変換テーブル中のラベルの重複をチェックする．同一名称のラベルが存在する場合はエラー(Duplicated label name)とする．ただし loop ラベルだけは例外とする．

(5)Path3: TIP プログラムの解析とメモリへの格納

A. プログラム行数(max_step_no)に基づいて，TIP プログラム格納領域を確保する．メモリが確保できない場合はエラー(Can not allocate memory for TIP program)とする．
B. 中間言語ファイル(FORMAL.TMP)を（読み込み/テキスト形式で）開く．
C. 変数使用履歴テーブルを確保し，ゼロクリアする．
D. 文字列テーブルを確保し，NULL クリアする．string.num をゼロクリアする．
E. プログラムステップ番号を０にリセットする．
F. 以下，END 行まで繰り返す．
   ① １行分の文字列を中間言語ファイル(FORMAL.TMP)から読み込む．
   ② 入力した文字列をコマンド文字列，引数文字列１，...，引数文字列ｎ（ｎは未定）に分割する．
   ③ コマンド文字列が存在しない(=NULL，空行)場合は①に戻り，次の行を解析する．
   ④ コマンド文字列を解析し，各コマンド毎に引数をチェックする．コマンド文字列が予約語一覧に存在しない場合はエラーとする(Unknown command)．必要な引数が存在しない，適当な引数ではない（変数の種類が異なるなど）場合はエラー(Wrong argument)とする．ただし変数の演算コマンドの場合は，演算子が引数文字列１に存在するので，それをもって解析を行う．
      - ジャンプ系コマンドを発見したら，そのラベル名と一致するラベルをラベル変換テーブルから探す．存在しない場合はエラー(Wrong label name)とする．ラベル名が loop だった場合は，ラベル変換テーブルをスキャンし，現在のプログラムステップ番号よりもプログラムステップ番号が小さく，かつ最も近い loop ラベルを探す．
      - 文字列関連のコマンド(MESSAGE)の場合は，文字列を文字列テーブルにコピーし，string.num を＋１する．max_num よりも大きいならばエラー．
      - 変数が使用されている場合は，変数使用履歴テーブルの対応するビットを１にする．汎用変数=0x01,，ポジション変数=0x02，関節角度変数=0x04，関節トルク変数=0x08．さらに値が代入された場合は，0x011，0x012，0x014，0x018 を OR する．
   ⑤ プログラムステップ番号を＋１する．
E. 中間言語ファイル(FORMAL.TMP)を閉じる．読み込みは成功．
F. END コマンドが存在しなかった場合はエラー(Not exist END)とする．
G. 変数使用履歴テーブルを元にして変数領域を確保する．
   - 変数使用履歴テーブルのビットを調べて，利用されているが値の代入されていない変数が存在するかチェックする．(Possible use of Variable before definition)．ただし構造体変数の各メンバ単位での代入し忘れのチェックはしない．
   - 各変数型の使用個数を調べ， 各変数型毎に必要な数だけ変数番号変換テーブルを確保する．
   - 各変数型の変数番号変換テーブルを初期化(number に変数番号，value に初期値=99999999.9)する．初期化が終了したら，変数使用履歴テーブル領域を開放する．
   -

(6) path4: 制御構文のチェックと step_no の更新

（１） メモリに格納された TIP プログラムを検索して，制御構文（IF_THEN，ELSE，ENDIF，WHILE，WEND，REPEAT，UNTIL，FOR，NEXT，SUBROUTINE，SUBEND）の数を数える．
（２） 制御構文チェック用のスタック領域を確保する．
（３） メモリ中の TIP プログラムを検索しながら，構造のチェックと step_no（ジャンプ先：arg_value）を更新する．
（４） ジャンプ命令（GOTO,GOSUB,IF_GOTO,IF_GOSUB）が内側の制御ループ

（IF_THEN,WHILE,REPEAT,FOR)に入り込まないこと（外側に抜けるのは構わない），SUBROUTINE 内に外部から入り込むことがないのをチェックする．

(7)自動運転開始(S)でプログラムをプログラムステップ番号０から１行ずつ実行して行く．

(8)変数の値が参照される際に，値が事前に代入されていない（＝初期値）場合はエラー(Possible use of　Variable before definition)とする．

(9)END コマンドに到達したら，自動運転を終了して手動モードに遷移する．

（注意事項）

- エラーが発生した場合は，ファイルを閉じ，即座に非常停止モードに遷移する．
- 一時停止／再開，自動運転終了，終了予約などのオペレーションは Auto プログラム実行時と完全に同一とする．ただし KEY_IN コマンド実行時は除く．

4-3 ラベル

条件分岐などのジャンプ先名						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（ラベル名）：					
種別	汎用					
引数	なし					
機能	ジャンプ系コマンドのジャンプ先として用いる.					
注意	1. ラベルは，１つのコマンドとして解釈される．従って，同一行にラベルとコマンドを記述することはできない. 2. ラベル名は一つの TIP プログラム内で重複して宣言してはいけない．ただし LOOP: ラベル(GOTO 系のみ．GOSUB 系では使用不可)は除く. 3. 予約語は除く. 4. ラベル名は半角英数字とし，大文字／小文字は区別しない. 5. ラベル名の長さは TIP_LABEL_MAX_CHAR(３２文字[byte])以内とする. 6. 必ず：（コロン）を最後に付ける.					
例	L1: GOTO L1:					
See	**GOTO, IF**					
メモリ格納形式	**Cmd**	LABEL_NAME				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

４-４ 変数の種類

変数の種類には，

1. 汎用型変数　　：DEF_VAR
2. 座標型変数　　：DEF_POS
3. 関節型変数　　：DEF_JOINT

の三種類[4]が存在する．

それぞれ値は double 型，変数名は半角英数大文字である．

座標型変数は３つのメンバ，関節型変数は各ロボットにおける１本脚（あるいは指）あたりの最大関節数（TitanVIII，HiroshimaHand の場合は３）のメンバを持つ構造体変数である．

- ポジション変数：

POS.X　＝　脚先Ｘ座標［mm］
POS.Y　＝　脚先Ｙ座標［mm］
POS.Z　＝　脚先Ｚ座標［mm］

- 関節型変数：

JOINT.J１　＝　関節１の角度［deg］
JOINT.J２　＝　関節２の角度［deg］
JOINT.J３　＝　関節３の角度［deg］
・・・

変数は，プログラム実行前に初期値（TIP_DEFAULT_VALUE_DOUBLE=－99999999.9）に初期化されている．右辺に用いる際に値が事前に代入されていない場合は実行時(Runtime)エラーとする．またコマンドの引数として用いられる場合も，一部のコマンド（値を代入するコマンド：教示系コマンド，AD_IN など）を除き，初期値の場合は実行時(Runtime)エラーとする．

TIP プログラム内で使用される変数は，TIP プログラム内で全て宣言する必要がある．変数の名前は自由である．ただし，変数名の長さは 32byte（半角３２文字，全角使用不可）[5]以内とする．

各 TIP コマンド名，特殊変数，特殊定数は予約語とし，変数名としての使用はできない．

（例）

```
DEF_VAR          VAR10, SW, FOOT1
DEF_POS          POS10, F1_TOP, F2_TOP, F3_TOP
DEF_JOINT        STIFFNESS_F1, INITIAL_ANGLE_F1

KEY_IN  VAR10
F1_TOP.X = F2_TOP.X + F3_TOP.X
STIFFNESS_F1.J1 = VAR10 / 100.0
```

（付記）

・ 将来的には，BOOLEAN 型の変数を作って，スイッチ類の入力(Digital Input :DI)をサポート予定．ただし，汎用変数でも現状では対応できるので，そちらを使用する．

---

[4] Ver.3 以前では，VAR, POS, AGL, TRQ の４種類の変数があった．しかし，AGL,TRQ は JOINT 型変数であり，使い分ける用途も存在しないため，効率を良くし，コードを小型化するために，統合する．更に自由な変数名への対応も同時に行った．

[5] ラベルおよび変数名の全角対応も考えたが，メリットよりもデメリットの方が多いため見送る．要望が多いならば考慮可能である．

４-５ 特殊変数と特殊定数

以下の変数および定数は予約されているので，変数名として使用することはできない.

４-５-１ 特集変数

変数名	説明
VAR_KEY	キー入力（KEY_IN）で引数を指定されなかった場合に使用される.

４-５-２ 特殊定数

定数名	説明
CHAR_0～CHAR_9	'0'から'9'のアスキーコード
CHAR_A～CHAR_Z	'A'から'Z'のアスキーコード
CHAR_SPC	' '(スペースバー）のアスキーコード

５　変数の演算と代入

演算子として,
　1.四則演算（＋，－，＊，／）
　2.代入（＝）
　3.整数化 (INT)
　4.絶対値 (ABS)
　5.平方根 (SQRT)
　6.三角関数１ (SIN,COS,TAN)
　7.三角関数２(ATAN,ATAN2,ASIN,ACOS)
　8.ベクトル演算 (NORM, UNIT_VECT)
が用意されている.

全ての算術演算に以下の代入演算が許される.

1.　（変数１）　＋＝　（変数２）　：変数１＝変数１＋変数２
2.　（変数１）　－＝　（変数２）　：変数１＝変数１－変数２
3.　（変数１）　＊＝　（変数２）　：変数１＝変数１＊変数２
4.　（変数１）　／＝　（変数２）　：変数１＝変数１／変数２

以下,数値演算に関する共通の注意事項である.

- 式の左辺は必ず変数であること．もし左辺が定数の場合はエラー(Invalid equation format)とする．また，－（負号）付きの変数も不可とする（-VAR01 = ...）.
- 異なる変数間の代入は不可とする．ただし構造体変数のメンバと汎用変数間の代入は可とする.
- 演算結果を自分自身に書き込むことも可能.

[例]

```
POS01        = POS02              AGL01    = AGL02
VAR01        = VAR01 + 1.0        VAR01    += VAR01 + 1.0
VAR01        = INT VAR03          VAR01    -= INT VAR03
POS01.X      = SQRT VAR02         POS01.X  *= SQRT VAR02
TRQ03.J1     = ABS TRQ02.J1       TRQ03.J1 /= ABS TRQ02.J1
VAR01        = SIN VAR02          VAR01    = ASIN VAR02
VAR02        = COS POS01.X        VAR02    = ACOS POS01.X
AGL01.J3     = ATAN VAR02         TRQ02.J2 += ATAN2 VAR02 0.1
VAR01        = NORM POS02         POS02    = UNIT_VECT POS02
AGL01.J1     = DEG2RAD VAR02      VAR02    = RAD2DEG AGL.J1
```

（間違った例）

```
VAR02        = SQRT VAR01 + VAR03 ： 複数の演算を組み合わせている.
POS01        = NORM POS02
AGL01        = DEG2RAD AGL02
```

５-１ 四則演算

変数および定数の四則演算						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（変数）＝（変数｜定数）　四則演算子　（変数｜定数）					
種別	汎用					
引数						
機能	加算（＋），減算（－），乗算（＊），除算（／）					
注意	右辺は必ず変数であること. 構造体変数と汎用変数あるいは構造体メンバ変数間の四則演算はできない.					
例	VAR2　= VAR3　+ VAR4 POS1.X = VAR3　- VAR4 VAR2　= 100.0 * VAR5 VAR2　+= POS1.X / VAR6					
See						
メモリ格納形式	Cmd	EQUAL_ADD / SUB / MULT /DIV				
	Target	NO_TARGET				
	ArgType1	enum argument_typeT;　左辺の種類（定数不可）				
	ArgType2	enum argument_typeT;　右辺第一引数の種類				
	ArgType3	enum argument_typeT;　右辺第二引数の種類				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の値（変数の時は添え字）				
	Value3	右辺第二引数の値（変数の時は添え字）				

５-２　その他の算術演算

５-２-１　代入および基本算術演算（＝，ＩＮＴ，ＡＢＳ，ＳＱＲＴ）

1. 代入（＝）　　：ＥＱＵＡＬ
2. 整数化(INT)　：ＥＱＵＡＬ＿ＩＮＴ
3. 絶対値(ABS)　：ＥＱＵＡＬ＿ＡＢＳ
4. 平方根(SQRT)　：ＥＱＵＡＬ＿ＳＱＲＴ

整数化（ＩＮＴ）は，小数点以下の切り捨てを行う．Ｃ言語の floor()関数に相当する．
平方根（ＳＱＲＴ）は，引数が負の場合はエラー(Domain Error)を発生する．

代入および基本算術演算（= / INT / ABS / SQRT）						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（変数）＝（変数｜定数），（変数）＝（INT/ABS/SQRT）（変数｜定数）					
種別	汎用					
引数						
機能	基本的な算術演算					
注意	代入（＝）以外は，構造体型変数に対して使用することはできない．					
例	VAR01 = 1.0　　：定数を代入することもできる． POS02 = POS03　　：構造体型の代入は可能． VAR02 = INT AGL02.J1　：メンバ型での利用は， POS01.X = SQRT VAR02　：右辺，左辺共に可能．					
See						
メモリ格納形式	**Cmd**	EQUAL / EQUAL_(MINUS / INT / ABS / SQRT)				
	Target	NO_TARGET, または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT;　左辺の種類（定数不可）				
	ArgType2	enum argument_typeT;　右辺の種類				
	ArgType3	enum argument_typeT;　ARG_NONE				
	Value1	左辺の変数の添え字				
	Value2	右辺の値（変数の時は添え字）				
	Value3	0.0				

５-２-２　三角関数（SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2）

以下の三角関数が準備されている．このうち，ATAN2 コマンド以外は右辺第２引数を持たない.角度は全て度([deg])で計算される．[6]　三角関数２系のコマンドの左辺は，－180[deg]から+180[deg]に正規化されて出力される.

1.三角関数１　：　ＥＱＵＡＬ＿ＳＩＮ，ＥＱＵＡＬ＿ＣＯＳ，ＥＱＵＡＬ＿ＴＡＮ
2.三角関数２　：　ＥＱＵＡＬ＿ＡＴＡＮ，ＥＱＵＡＬ＿ＡＴＡＮ２，
ＥＱＵＡＬ＿ＡＳＩＮ，ＥＱＵＡＬ＿ＡＣＯＳ

SIN / COS / TAN / ASIN / ACOS / ATAN / ATAN2						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	(変数) = SIN / COS / TAN / ASIN / ACOS / ATAN　(定数|変数)					
種別	汎用					
引数	SIN, COS, TAN の右辺に与える角度，ATAN, ASIN, ACOS コマンドの左辺に代入される角度の単位は度([deg])である．ASIN, ACOS, ATAN の引数が，-1.0 から 1.0 の範囲内にない場合，エラー(Domain Error)を発生する.					
機能	三角関数演算を行う. ATAN2 arg1 arg2 は，ATAN(arg1/arg2)					
注意						
例	VAR02　= SIN VAR03 VAR02　= COS 30.0 VAR02　= TAN AGL02.J1 VAR02　= ASIN 0.5 AGL02.J1 = ACOS VAR03 TRQ02.J2 = ATAN VAR03 VAR02　= ATAN2 VAR03 VAR04					
See						
メモリ格納形式	**Cmd**	EQUAL_SIN, EQUAL_COS, EQUAL_TAN EQUAL_ASIN, EQUAL_ACOS, EQUAL_ATAN				
	Target	NO_TARGET, または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT;　左辺の種類（定数不可）				
	ArgType2	enum argument_typeT;　右辺第一引数の種類				
	ArgType3	enum argument_typeT;　ARG_NONE				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の値（変数の時は添え字）				
	Value3	0.0				
メモリ格納形式(ATAN2)	**Cmd**	EQUAL_ATAN2				
	Target	NO_TARGET				
	ArgType1	: enum argument_typeT;　左辺の種類（定数不可）				
	ArgType2	enum argument_typeT;　右辺第一引数の種類				
	ArgType3	enum argument_typeT;　右辺第二引数の種類				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の値（変数の時は添え字）				
	Value3	右辺第二引数の値（変数の時は添え字）				

[6] ANGLE コマンドなど，角度制御系のコマンドの引数の単位が度([deg])のため.

５-２-３　単位変換（DEG2RAD, RAD2DEG）

DEG2RAD / RAD2DEG						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	(変数) = DEG2RAD / RA2DEG (定数 | 変数)					
種別	汎用					
引数	右辺は変数，あるいは定数，左辺は変数．構造体型は不可（メンバ型は可）					
機能	DEG2RAD は度([deg])から[rad]へ，RAD2DEG は[rad]から[deg]へ値を変換する．					
注意						
例	VAR02 = DEG2RAD VAR03 AGL02.J1 = DEG2RAD AGL03.J2 AGL02.J1 = RAD2DEG VAR03					
See						
メモリ格納形式	Cmd	EQUAL_DEG2RAD, EQUAL_RAD2DEG				
	Target	NO_TARGET, または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT; 左辺の種類（定数不可）				
	ArgType2	enum argument_typeT; 右辺第一引数の種類				
	ArgType3	enum argument_typeT; ARG_NONE				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の値（変数の時は添え字）				
	Value3	0.0				

５-２-４　ベクトル演算（NORM, UNIT_VECT, ROT_X/Y/Z）

NORM						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（変数） ＝ NORM （座標型変数または関節型変数：構造体のみ）					
種別	汎用					
引数	右辺は座標型変数または関節型変数，左辺は汎用型変数					
機能	ユークリッドノルム（二乗平均）を求める． Sqrt($X^2 + Y^2 + Z^2$)，または Sqrt($J1^2+J2^2+J3^2+....$) [7]					
注意						
例	VAR02 = NORM POS01 POS01.X = NORM POS02		JE = J1 - J2 J_ERR = NORM JE			
See						
メモリ格納形式	Cmd	EQUAL_NORM				
	Target	NO_TARGET，または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT； 左辺の種類（VAR）				
	ArgType2	enum argument_typeT； 右辺の種類（座標型変数: POS,関節型変数: JNT）				
	ArgType3	enum argument_typeT； ARG_NONE				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の添え字				
	Value3	0.0				

UNIT_VECT						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（構造体型ポジション変数） ＝ UNIT_VECT （構造体型ポジション変数）					
種別	汎用					
引数	右辺，左辺ともにポジション変数（構造体のみ）					
機能	単位ベクトル化					
注意						
例	POS01 = UNIT_VECT POS02					
See						
メモリ格納形式	Cmd	EQUAL_UNIT_VECT				
	Target	NO_TARGET，または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT； 左辺の種類（ポジション変数: POS）				
	ArgType2	enum argument_typeT； 右辺の種類（ポジション変数: POS）				
	ArgType3	enum argument_typeT； ARG_NONE				
	Value1	左辺の変数の添え字				
	Value2	右辺の変数の添え字				
	Value3	0.0				

[7] 例えば，関節の変化量(JE = J1 - J2)の二乗平均を求めることで，関節が動いているか，（ほぼ）静止しているか，を判断することができる．

ROT_X, ROT_Y, ROT_Z						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	（構造体型ポジション変数 1） = ROT_X/Y/Z （構造体型ポジション変数 2）（回転角度）					
種別	汎用					
引数	右辺，左辺第一引数ともにポジション変数（構造体のみ） 回転角度： 汎用変数または即値：単位は[deg]					
機能	（構造体型ポジション変数 2）を X/Y/Z 軸回りに（回転角度）だけ回転させる.					
注意	回転はロボット座標系の X, Y, Z 軸に対して行われる.					
例	POS01 = ROT_X POS02 10					
See						
メモリ格納形式	Cmd	EQUAL_ROT_X, EQUAL_ROT_Y, EQUAL_ROT_Z				
	Target	NO_TARGET, または CAL_LEFT_ADD, SUB, MULT, DIV				
	ArgType1	enum argument_typeT; 左辺の種類（ポジション変数: POS）				
	ArgType2	enum argument_typeT; 右辺第一引数の種類（ポジション変数: POS）				
	ArgType3	enum argument_typeT; 右辺第二引数の種類（CONSTANT または VAR）				
	Value1	左辺の変数の添え字				
	Value2	右辺第一引数の変数の添え字				
	Value3	即値，あるいは右辺第二引数の変数の添え字				

６-１　関係演算子（==, <>, <, >, <=, >=）

関係演算子は，条件文(IF)においてのみ使用可能である．
また構造体変数間の関係演算子は使用できない[8]．

一般形は，

｜汎用変数		｜汎用変数
｜構造体変数（メンバ表記）	（関係演算子）	｜構造体変数（メンバ表記）
｜定数		｜定数

である．

使用できる関係演算子は以下の６種類である．

1. ＝＝　　：右辺と左辺が等しい
2. ＜＞　　：右辺と左辺が等しくない
3. ＜　　：右辺が左辺より大きい
4. ＞　　：右辺が左辺より小さい
5. ＜＝　　：右辺が左辺以上である
6. ＞＝　　：右辺が左辺以下である

［例］

```
IF VAR01 == VAR02 GOTO L1:
IF VAR01 <> 0.0 GOTO L1:
IF 2.5 < VAR02 GOTO L1:
IF VAR02 > 2.5 GOTO L1:
IF POS01.X <= VAR02 GOTO L1:
IF AGL.J1 > TRQ.J2 GOTO L1:
```

［メモリ格納形式］

IF コマンド参照．

[8] どうしても比較したいのならば，３つの IF 文を連ねるなどの方法を取ること．そもそも必要になる状況が想像できない

６-２　イベントフラグ

特定の条件が揃ったときにＯＮになるフラグで，主に IF_GOTO/IF_GOSUB の条件式で用いられる．
現在,

- HALT_REQUEST
- KB_HIT
- INPOS_F? / INPOS_ALL
- NEAR_F? / NEAR_ALL
- SPECIAL_FLAGn

が用意されている.あとは適時追加．イベントフラグの前に！を付けることで論理を反転することができる．

例） IF !KB_HIT GOTO L1:

（注意）　！とイベントフラグ名の間にスペースを入れてはいけない．

６-２-１ HALT_REQUEST

HALT_REQUEST　：　サイクル停止要求のチェック						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
種別	汎用					
機能	メインプログラムからサイクル停止（区切りの良いところで停止しろ）の指示があったときに真となる．					
注意						
例	IF　HALT_REQUEST GOTO FINISH: サイクル停止要求がメインプログラムから発せられているならばFINISH:へジャンプする．					
See						

６-２-２ KB_HIT

KB_HIT　：　キー入力のチェック						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
種別	汎用					
機能	キーボード（またはRS232C）からの入力があったらＴＲＵＥとなる．					
注意						
例	IF　KB_HIT GOSUB KEY_CHECK: もしキーボードが押されたならば，KEY_CHECK へジャンプする．． キーボードが“いま現在押されているか”ではなく，“キーボードバッファに文字が溜まっているかをチェック”する．事前に，FLUSH_BUF コマンドでキーボードバッファを空にしておくと良い.KEY_IN コマンドは実行後，キーボードバッファを完全に空にするが，KB_HIT はキーボードバッファを一切いじらない．したがって，改めて別の KB_HIT による条件チェックを行う場合には，意図的に FLUSH_BUF を行う必要がある．					
See	**FLUSH_BUF, KEY_IN**					

6-2-3 INPOS_F?／INPOS_J?／INPOS_ALL

INPOS_F?／INPOS_J?／INPOS_ALL　：　目標位置／角度到達をチェックする						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
種別	汎用					
機能	・位置制御 PTP/REL_PTP COMPLIANCE_POSITION/REL_COMPLIANCE_POSITION ・　角度制御 PTP_ANGLE, REL_PTP_ANGLE COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE) されている脚／関節が目標位置に到達している場合に TRUE となる. **（チェック対象）** **IN_POS_F?:** 指定した脚先（Ｆ？：Ｆ１，Ｆ２，Ｆ３，Ｆ４）に含まれる位置制御／角度制御された個所が全て目標位置／目標角度に到達している場合に TRUE となる. **IN_POS_J?:** 指定した関節が目標角度に到達している場合に TRUE となる. **IN_POS_ALL:** 全ての脚のうち，位置制御／角度制御されている個所が全て目標位置／角度に到達している場合に TRUE となる.					
注意	位置制御されている脚／関節が存在しない場合は常に TRUE を返す[9]. コンプライアンス制御された関節も位置／角度制御の対象と見なされる点に注意. 特定の位置制御された脚／関節のみをチェックしたい場合は各関節を指定した IF 文を連ねる.					
例	L1: IF　INPOS_F1 GOTO L2: GOTO L1: もし，Ｆ１が目標位置に到達しているならば真になる．指定された足が位置制御モードではない時は，TRUE を返す.					
See	**PTP, REL_PTP, SET_JUST_POSITION,** **COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION,** **WAIT_INPOS** **NEAR_F?/J?/ALL**					

6-2-4 NEAR_F?／NEAR_J?／NEAR_ALL

NEAR_F?／NEAR_J?／NEAR_ALL　：　目標位置／角度近傍到着をチェックする						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
種別	汎用					
機能	INPOS_F?／J?／ALL の目標位置／角度近傍版である.					
注意						
例	L1: IF　NEAR_ALL GOTO L2: GOTO L1: もし，位置制御／角度制御されている脚／関節が全て目標位置近傍に到達しているならば真になる．位置制御／角度制御モードの脚／関節が存在しない時は，FALSE のままである.					
See	**INPOS_F?／INPOS_J?／INPOS_ALL**					

9 これは基本的に TIP プログラマーのミスである.

６-２-５ SPECIAL_FLAGn

SPECIAL_FLAGn　：　外部コマンドからの汎用フラグをチェックする						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
種別	汎用					
機能	SPECIAL_FLAG0 から SPECIAL_FLAG9 までの１０個が用意されている．TIP プログラム起動時には，全て偽にリセットされている．外部プログラムから自由に ON/OFF できる．					
注意						
例	L1: IF SPECIAL_FLAG0 GOTO L2: GOTO L1: SPECIAL_FLAG0 が真になったらＬ２へジャンプする．					
See	SPECIAL					

7　コマンドリファレンス

コマンドには，大きく分けて

1. 移動系コマンド
2. 速度制御系コマンド
3. 力制御系コマンド
4. WAIT 系コマンド
5. 教示系コマンド
6. ジャンプ系コマンド
7. パラメータ調整系コマンド
8. 入出力系コマンド
9. その他

が存在する．それ以外にもラベルと変数の演算と代入が可能である．

7-1 移動系コマンド

7-1-1　PTP

<table>
<tr><th colspan="7">PTP　：　足先位置の絶対位置制御</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">PTP　(HS)　(脚識別子)　(R|A)　(x 座標)　(R|A)(y 座標)　(R|A)(z 座標)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">HS　　：高速に移動することを指示する.
脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
(HiroshimaHand は，J13，J23，J33 の位置制御も可能)
R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する.
A は省略できる.
目標値：定数あるいは変数．定数の場合は，小数点および＋記号を省略可能.</td></tr>
<tr><td>機能</td><td colspan="6">本コマンドを解釈後，指定された脚および関節に対する robot 構造体の目標値および制御モードを変更，即座に反映される（＝ロボットは目標値に向かって動き始める）．プログラムステップ番号は目標値への到達を待たずに次のステップに移る.</td></tr>
<tr><td>注意</td><td colspan="6">HiroshimaHand の場合，z 座標は無視される.</td></tr>
<tr><td>例</td><td colspan="6">PTP F1 100.0 -100.0 0.0
：脚 F1 を(100.0, -100.0, 0.0)へ動かす
PTP F4 R 10.0 R 20.0 R -10.0
：脚 F4 を現在の位置から相対的に(10.0, 20.0, -10.0)の位置へ動かす.
PTP F3 POS01
：脚 F3 を(POS01.X, POS01.Y, POS01.Z)へ動かす.
PTP F3 POS01.X POS01.Y R VAR01
：脚 F3 を(POS01.X, POS01.Y, 現在の Z 座標+VAR01)へ動かす.</td></tr>
<tr><td>See</td><td colspan="6">PTP_ANGLE, ALL_STOP, ALL_STOP_RELEASE,
WAIT_INPOS, WAIT_NEAR, TEACH</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">PTP / PTP_HS</td></tr>
<tr><td>Target</td><td colspan="5">脚識別子(F1, F2, F3, F4: 省略不可)
HiroshimaHand の場合は，関節識別子(J13, J23, J33)も可</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT;　第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT;　第二引数の種類（無い時は ARG_NONE</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT;　第三引数の種類（無い時は ARG_NONE)</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字)</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
</table>

7-1-2　REL_PTP

<table>
<tr><th colspan="7">REL_PTP　：　足先の相対位置制御</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">REL_PTP　(HS)　(脚識別子)　(x 座標)　(y 座標)　(z 座標)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">HS　　：高速に移動することを指示する.
脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
（HiroshimaHand は，J13，J23，J33 の位置制御も可能）
目標値：定数あるいは変数．定数の場合は，小数点および＋記号を省略可能.</td></tr>
<tr><td>機能</td><td colspan="6">相対座標による脚先位置制御を行う．PTP コマンドの相対座標版．PTP コマンドでも相対座標の指示は可能だが，間違いやすいので．本コマンドを解釈後，指定された脚および関節に対する robot 構造体の目標値および制御モードを変更，即座に反映される（＝ロボットは目標値に向かって動き始める）．プログラムステップ番号は目標値への到達を待たずに次のステップに移る.</td></tr>
<tr><td>注意</td><td colspan="6">REL_PTP コマンドは，目標位置誤差の累積を防ぐために，”現在の位置”に対する相対位置ではなく，”現在の目標位置”に対する相対位置へ移動するように実装される．したがって，いま現在，ある目標位置に向かって移動している脚に対して REL_PTP を行う場合は，一旦，REL_PTP ? 0.0 0.0 0.0 を行って停止した上で，新たな相対位置指令を与えるのが望ましい．そのためには，REL_PTP の引数が(0.0, 0.0, 0.0)の時のみ”その場停止”するようにロボットプログラムが実装されている必要がある.
HiroshimaHand の場合，z 座標は無視される.</td></tr>
<tr><td>例</td><td colspan="6">REL_PTP F1 100.0 -100.0 0.0
：脚 F1 を現在の位置から相対的に(100.0, -100.0, 0.0)動かす
REL_PTP F3 POS01
：脚 F3 を現在の位置から相対的に(POS01.X, POS01.Y, POS01.Z)動かす.
REL_PTP F3 POS01.X POS01.Y VAR01
：脚 F3 現在の位置から相対的に(POS01.X, POS01.Y, VAR01)動かす.</td></tr>
<tr><td>See</td><td colspan="6">PTP, PTP_ANGLE, ALL_STOP, ALL_STOP_RELEASE,
WAIT_INPOS, WAIT_NEAR, TEACH</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">REL_PTP / REL_PTP_HS</td></tr>
<tr><td>Target</td><td colspan="5">脚識別子(F1, F2, F3, F4: 省略不可)
HiroshimaHand の場合は，関節識別子(J13, J23, J33)も可</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT;　第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT;　第二引数の種類（無い時は ARG_NONE）</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT;　第三引数の種類（無い時は ARG_NONE）</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字）</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
</table>

7-1-3　PTP_ANGLE

<table>
<tr><th colspan="7">PTP_ANGLE　：　関節角度の絶対角度制御</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">PTP_ANGLE　(HS)　(脚/関節識別子)　(R|A)　(目標値１). . .　(R|A)(目標値 3)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">HS　　：高速に移動することを指示する.
脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３
R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する.
A は省略できる.
目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能.</td></tr>
<tr><td>機能</td><td colspan="6">本コマンドを解釈後，指定された脚および関節に対する robot 構造体の目標値および制御モードを変更，即座に反映される（＝ロボットは目標値に向かって動き始める）．プログラムステップ番号は目標値への到達を待たずに次のステップに移る.</td></tr>
<tr><td>注意</td><td colspan="6"></td></tr>
<tr><td>例</td><td colspan="6">PTP_ANGLE F1 R 10.0 R -10.0 R 0.0
： 脚 F1 の関節 J11 を+10[deg]， J12 を-10[deg]， J13 を+0.0[deg]動かす
PTP_ANGLE J23 10.0
： 脚 2 の関節 J23 を 10[deg]へ動かす.
PTP_ANGLE F2 AGL01
：脚 2 の各関節を(ANG01.J1， ANG02.J2， ANG02.J3)へ動かす.
PTP_ANGLE F2 AGL02.J1 AGL03.J2 VAR03
：脚 2 の各関節を(AGL01.J1， AGL03.J2， VAR03)へ動かす.</td></tr>
<tr><td>See</td><td colspan="6">PTP, REL_PTP_ANGLE, ALL_STOP, ALL_STOP_RELEASE,
WAIT_INPOS, WAIT_NEAR, TEACH,
COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">PTP_ANGLE / PTP_ANGLE_HS</td></tr>
<tr><td>Target</td><td colspan="5">脚／関節識別子(F1～F4，J11～J43: 省略不可)</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT;　第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT;　第二引数の種類（無い時は ARG_NONE)</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT;　第三引数の種類（無い時は ARG?NONE)</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字)</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字，無いときは 0.0)</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字，無いときは 0.0)</td></tr>
</table>

7-1-4　REL_PTP_ANGLE

<table>
<tr><th colspan="7">REL_PTP_ANGLE　：　関節角度の相対角度制御</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">REL_PTP_ANGLE　(HS)　(脚/関節識別子) (目標値１)...(目標値 3)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">HS　　：高速に移動することを指示する.
脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３
目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能.</td></tr>
<tr><td>機能</td><td colspan="6">相対角度による関節角度制御を行う. PTP_ANGLE コマンドの相対座標版. PTP_ANGLE コマンドでも相対角度の指示は可能だが，間違いやすいので. 本コマンドを解釈後，指定された脚および関節に対する robot 構造体の目標値および制御モードを変更，即座に反映される（＝ロボットは目標値に向かって動き始める）. プログラムステップ番号は目標値への到達を待たずに次のステップに移る.</td></tr>
<tr><td>注意</td><td colspan="6"></td></tr>
<tr><td>例</td><td colspan="6">REL_PTP_ANGLE F1 10.0 -10.0 0.0
: 脚 F1 の関節 J11 を+10[deg], J12 を-10[deg], J13 を+0.0[deg]動かす
REL_PTP_ANGLE J23 10.0
: 脚 2 の関節 J23 を+10[deg]動かす.
REL_PTP_ANGLE F2 AGL01
：脚 2 の各関節を(+ANG01.J1, +ANG02.J2, +ANG02.J3)動かす.
REL_PTP_ANGLE F2 AGL02.J1 AGL03.J2 VAR03
：脚 2 の各関節を(+AGL01.J1, +AGL03.J2, +VAR03)動かす.</td></tr>
<tr><td>See</td><td colspan="6">PTP, PTP_ANGLE, ALL_STOP, ALL_STOP_RELEASE,
WAIT_INPOS, WAIT_NEAR, TEACH,
COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">REL_PTP_ANGLE / REL_PTP_ANGLE_HS</td></tr>
<tr><td>Target</td><td colspan="5">脚／関節識別子(F1～F4, J11～J43: 省略不可)</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT;　第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT;　第二引数の種類（無い時は ARG_NONE）</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT;　第三引数の種類（無い時は ARG?NONE)</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字）</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字, 無いときは 0.0）</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字, 無いときは 0.0）</td></tr>
</table>

7-1-5　ALL_STOP

ALL_STOP　：　全関節の動作を一時中断						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	ALL_STOP					
種別	汎用					
引数	なし					
機能	全関節の動作を一時的に中断する．ただしモータドライバへの出力を停止（0[V]）するのではなく，現在の角度を維持するように，一時的に角度制御される．その際に robot 構造体の制御モードや目標値は変更せずに，robot_controller()の中で例外処理として処理する．利用目的としては， 1.　その場停止（KEY_IN を待つ間など） 2.　全ての目標値を確実に設定し終えてから同時に動作を開始したい場合 3.　ファイルへのアクセス(OUTPUT_VARIABLE など) などが考えられる． この機能は，力制御系コマンド実行時にも有効である．					
注意						
例	ALL_STOP					
See	ALL_STOP_RELEASE					
メモリ格納形式	Cmd	ALL_STOP				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-1-6　ALL_STOP_RELEASE

ALL_STOP_RELEASE　：　全関節一時中断を解除						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	ALL_STOP_RELEASE					
種別	汎用					
引数	なし					
機能	ALL_STOP により一時的に止められていた制御を再開する. ALL_STOP されていないならば意味はない.					
注意						
例	ALL_STOP_RELEASE					
See	ALL_STOP					
メモリ格納形式	Cmd	ALL_STOP_RELEASE				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-1-7　INITIAL_POSTURE

INITIAL_POSTURE　：　ロボットを初期姿勢に戻す						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	INITIAL_POSTURE					
種別	汎用					
引数	なし					
機能	ロボットを初期姿勢に戻すとともに，設定値やスイッチをデフォルトに戻す．ALL_STOP を行うことで脚先を現在の位置に停止させる．停止完了後，脚先座標目標値を初期姿勢に設定し，制御モードを脚先座標モードに切り替え，変更された設定値やスイッチをデフォルトに戻した上で ALL_STOP_RELEASE を実行し，初期姿勢に戻る．					
注意						
例	INITIAL_POSTURE					
See	ALL_STOP, ALL_STOP_RELEASE					
メモリ格納形式	Cmd	INITIAL_POSTURE				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

７-２　速度制御系コマンド

７-２-１　VEL_CONTROL

VEL_CONTROL　：　足先位置の速度制御						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	VEL_CONTROL　(脚識別子)　(R|A)　(目標値１). . .　(R|A)(目標値 3)					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４ R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する． A は省略できる． 目標値：脚先の速度(mm/sec)を指示する．定数あるいは変数．定数の場合は，小数点および＋記号を省略可能．					
機能	指示された速度で脚先を移動させる．(vx,vy,vz)形式か，あるいはポジション変数にて値を渡す．相対指令も可能．EMG_OFF してある場合は限界まで動いて止まる．それ以前に外部センサの入力(DIO，AD_IN)や WAIT 系コマンドによって止めるのが正しい．					
注意	HiroshimaHand の場合，ｚ成分は無視される．					
例	VEL_CONTROL F1 POS02 ：脚 F1 の脚先を速度(POS02.X，POS02.Y，POS02.Z)で動かす． VEL_CONTROL F1 R 10.0 A 0.0 A 0.0 ：脚 F1 の脚先を速度(+10.0，0.0，0.0)で動かす．					
See	PTP, PTP_ANGLE,, OMEGA_CONTROL					
メモリ格納形式	Cmd	VEL_CONTROL				
	Target	脚識別子(F1～F4: 省略不可)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類（無い時は ARG_NONE）				
	ArgType3	enum argument_typeT;　第三引数の種類（無い時は ARG_NONE）				
	Value1	第一引数の値（変数の時は添え字）				
	Value2	第二引数の値（変数の時は添え字，無いときは 0.0）				
	Value3	第三引数の値（変数の時は添え字，無いときは 0.0）				

７-２-２　OMEGA_CONTROL

<table>
<tr><th colspan="7">OMEGA_CONTROL　：　関節角度制御</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">OMEGA_CONTROL （脚/関節識別子） (R|A) （目標値１）... (R|A) (目標値 3)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３
R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する.
A は省略できる.
目標値：定数あるいは変数．定数の場合は，小数点および＋記号を省略可能.</td></tr>
<tr><td>機能</td><td colspan="6">本コマンドを解釈後，指定された脚および関節に対する robot 構造体の速度指令値および制御モードを変更，即座に反映される（＝ロボットは角速度で動き始める．もちろん角加速度制限は有効）．プログラムステップ番号は目標値への到達を待たずに次のステップに移る.</td></tr>
<tr><td>注意</td><td colspan="6">HiroshimaHand の場合， z 成分は無視される.</td></tr>
<tr><td>例</td><td colspan="6">OMEGA_CONTROL F1 R 10.0 R -10.0 R 0.0
： 脚 F1 の関節 J11 を+10[deg/sec]， J12 を-10[deg/sec]， J13 を+0.0[deg/sec]で動かす
OMEGA_CONTROL J23 10.0
： 脚 2 の関節 J23 を 10[deg/sec]で動かす.
OMEGA_CONTROL F2 AGL01
：脚 2 の各関節を(ANG01.J1， ANG02.J2， ANG02.J3)[deg/sec]で動かす.
OMEGA_CONTROL F2 AGL02.J1 AGL03.J2 VAR03
：脚 2 の各関節を(AGL01.J1， AGL03.J2， VAR03)[deg/sec]で動かす.</td></tr>
<tr><td>See</td><td colspan="6">PTP, ALL_STOP, ALL_STOP_RELEASE, WAIT_INPOS, WAIT_NEAR, TEACH</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">OMEGA_CONTROL</td></tr>
<tr><td>Target</td><td colspan="5">脚／関節識別子(F1～F4，J11～J43: 省略不可)</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT;　第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT;　第二引数の種類（無い時は ARG_NONE）</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT;　第三引数の種類（無い時は ARG_NONE）</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字）</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字，無いときは 0.0）</td></tr>
</table>

7-3　力制御系コマンド

7-3-1　TORQUE

TORQUE　：　関節トルク制御						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	TORQUE (脚/関節識別子) (R|A) (目標値１). . . (R|A)(目標値 3)					
種別	汎用					
引数	関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３ R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する． A は省略できる． 目標値：定数あるいは変数．定数の場合は，小数点および＋記号を省略可能．					
機能	本コマンドを解釈後，指定された脚および関節に対する robot 構造体の目標値および制御モードを変更，即座に反映される（＝ロボットは目標値に向かって動き始める）．					
注意	単位が[Nm]ではあるが，TitanVIII の場合はハードウェア上の制限により厳密な制御は実現不可能．なんとなくそれっぽいトルク制御ができれば上出来．					
例	TORQUE F1 R 10.0 R -10.0 R 0.0 ：脚 F1 の関節 J11 のトルクを+10[Nm]，J12 を-10[Nm]，J13 を+0.0[Nm]にする． TORQUE J23 10.0 ：脚 2 の関節 J23 のトルクを 10[Nm]にする． TORQUE F2 TRQ01 ：脚 2 の各関節トルクを(TRQ01.J1，TRQ02.J2，TRQ02.J3)にする． TORQUE F2 TRQ02.J1 TRQ03.J2 VAR03 ：脚 2 の各関節トルクを(TRQ01.J1，TRQ03.J2，VAR03)にする．					
See	**PTP，PTP_ANGLE, ALL_STOP, ALL_STOP_RELEASE, COMPLIANCE, NO_COMPLIANCE**					
メモリ格納形式	**Cmd**	TORQUE				
	Target	関節識別子(J11～J43: 省略不可)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類（無い時は ARG_NONE）				
	ArgType3	enum argument_typeT;　第三引数の種類（無い時は ARG_NONE）				
	Value1	第一引数の値（変数の時は添え字）				
	Value2	第二引数の値（変数の時は添え字，無いときは 0.0）				
	Value3	第三引数の値（変数の時は添え字，無いときは 0.0）				

7-3-2 FORCE

FORCE　：　力制御を行う						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	**FORCE （指/関節識別子） (R|A) （目標値１). . . (R|A)(目標値 3)**					
種別	汎用					
引数	指識別子　　：Ｆ１，Ｆ２，Ｆ３ 関節識別子　：Jn3 R|A　　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する. A は省略できる. 目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能.					
機能	指先（あるいは指定した関節）に対して，力制御を行う. 目標値は外界に対して指先（あるいは関節）に換算した接触力に対する反力の目標値. Titan の場合は接触力を計測する手段が無いので難しいので，実装の予定は無い.					
注意	HiroshimaHand の場合， z 成分は無視される.					
例	FORCE　F1 50.0 20.0 0.0 ：指 F1 の指先に対して(50.0, 20.0)[N]の力制御を行う. FORCE　J13 20.0 30.0 40.0 ：指 F1 の第 3 関節に対して(20.0, 30.0)[N]の力制御を行う. FORCE F1 POS3 ：指 F1 の指先に対して(POS3.x, POS3.y)[N]の力制御を行う. POS3.z は無視される.					
See	**COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION, PTP, REL_PTP, TORQUE**					
メモリ格納形式	Cmd	FORCE				
	Target	脚／関節識別子(F1～F3, J13,J23,J33: 省略不可)				
	ArgType1	enum argument_typeT; 第一引数の種類				
	ArgType2	enum argument_typeT; 第二引数の種類（無い時は ARG_NONE)				
	ArgType3	enum argument_typeT; 第三引数の種類（無い時は ARG?NONE)				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字，無いときは 0.0)				
	Value3	第三引数の値（変数の時は添え字，無いときは 0.0)				

7-3-3　COMPLIANCE_ANGLE

COMPLIANCE_ANGLE　：　関節の柔らかさを変える						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	COMPLIANCE_ANGLE (HS) (脚/関節識別子) (R|A) (目標値 1). . . (R|A) (目標値 3)					
種別	汎用					
引数	HS　：高速に移動することを指示する. 脚識別子　：Ｆ１，Ｆ２，Ｆ３，Ｆ４ 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３ 全脚指定　：ALL R|A　：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する. Aは省略できる. 目標値：定数あるいは変数．定数の場合は，小数点および＋記号を省略可能.					
機能	関節の柔らかさをコントロールする. 目標角度は絶対角度. 関節剛性は，SET_ANGLE_STIFFNESS コマンドで設定する．単位は[Nmm/deg]. Titan の場合は関節トルクを計測する手段が無いので難しい．例えば，モータへの出力電圧を小さくしてしまうなど.					
注意						
例	COMPLIANCE_ANGLE　J13 50.0 ：脚 F1 の関節 J13 のコンプライアンス制御目標角度を絶対角 50[deg]に設定する. COMPLIANCE_ANGLE　F1 20.0 30.0 40.0 ：脚 F1 の関節のコンプライアンス制御目標角度を それぞれ J11=20.0[deg]，J12=30.0[deg]，J13=40.0[deg]に設定する.					
See	REL_COMPLIANCE_ANGLE, NO_COMPLIANCE, SET_ANGLE_STIFFNESS, PTP_ANGLE, REL_PTP_ANGLE, TORQUE					
メモリ格納形式	Cmd	COMPLIANCE_ANGLE / COMPLIANCE_ANGLE_HS				
	Target	脚／関節識別子(F1～F4，J11～J43: 省略不可)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類（無い時は ARG_NONE)				
	ArgType3	enum argument_typeT;　第三引数の種類（無い時は ARG?NONE)				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字，無いときは 0.0)				
	Value3	第三引数の値（変数の時は添え字，無いときは 0.0)				

7-3-4 REL_COMPLIANCE_ANGLE

REL_COMPLIANCE_ANGLE　：　関節の柔らかさを変える						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	REL_COMPLIANCE_ANGLE (HS) (脚/関節識別子) (目標値１). . . (目標値 3)					
種別	汎用					
引数	HS　　：高速に移動することを指示する. 脚識別子　：Ｆ１，Ｆ２，Ｆ３，Ｆ４ 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３ 全脚指定　：ALL 目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能.					
機能	関節の柔らかさをコントロールする. 目標角度は相対角度. 関節剛性は，SET_ANGLE_STIFFNESS コマンドで設定する. 単位は[Nmm/deg]. Titan の場合は関節トルクを計測する手段が無いので難しい. 例えば，モータへの出力電圧を小さくしてしまうなど.					
注意						
例	REL_COMPLIANCE_ANGLE　J13 50.0 ：脚 F1 の関節 J13 のコンプライアンス制御目標角度を相対角 50[deg]に設定する. REL_COMPLIANCE_ANGLE　F1 20.0 30.0 40.0 ：脚 F1 の関節のコンプライアンス制御目標角度を それぞれ J11=20.0[deg], J12=30.0[deg], J13=40.0[deg]に設定する.					
See	COMPLIANCE_ANGLE, NO_COMPLIANCE, SET_ANGLE_STIFFNESS, PTP_ANGLE, REL_PTP_ANGLE, TORQUE					
メモリ格納形式	Cmd	REL_COMPLIANCE_ANGLE / REL_COMPLIANCE_ANGLE_HS				
	Target	脚／関節識別子(F1～F4, J11～J43: 省略不可)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類（無い時は ARG_NONE)				
	ArgType3	enum argument_typeT;　第三引数の種類（無い時は ARG?NONE)				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字, 無いときは 0.0)				
	Value3	第三引数の値（変数の時は添え字, 無いときは 0.0)				

7-3-5 COMPLIANCE_POSITION

<table>
<tr><th colspan="7">COMPLIANCE_POSITION ： 指先（など）にバネを設定する</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>×</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">COMPLIANCE_POSITION (HS) (指/関節識別子) (R|A) (目標値 1). . . (R|A)(目標値 3)</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">HS ：高速に移動することを指示する.
指識別子 ：Ｆ１，Ｆ２，Ｆ３
全指指定 ： ALL
関節識別子 ： Jn3
R|A ：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する.
A は省略できる.
目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能. 絶対座標.</td></tr>
<tr><td>機能</td><td colspan="6">目標位置からの距離に比例する力で引っ張るバネを指先（または指定の関節）に設定する.
バネの剛性は，SET_POSITION_STIFFNESS コマンドで設定する. 単位は[N/mm].
Titan の場合は接触力を計測する手段が無いので難しいので，実装の予定は無い.</td></tr>
<tr><td>注意</td><td colspan="6">HiroshimaHand の場合， z 座標は無視される.</td></tr>
<tr><td>例</td><td colspan="6">COMPLIANCE_POSITION F1 50.0 20.0 0.0
：指 F1 の指先と座標(50.0, 20.0)の間にバネを設定する.
COMPLIANCE_POSITION J13 20.0 30.0 40.0
：指 F1 の第 3 関節と座標(20.0, 30.0)の間にバネを設定する. 40.0 は無視される.
COMPLIANCE_POSITION J23 POS3
：指 F2 の第 3 関節と座標(POS3.x, POS3.y)の間にバネを設定する. POS3.z は無視される.</td></tr>
<tr><td>See</td><td colspan="6">REL_COMPLIANCE_POSITION, NO_COMPLIANCE,
SET_POSITION_STIFFNESS,
PTP, REL_PTP, TORQUE</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">COMPLIANCE_POSITION / COMPLIANCE_POSITION_HS</td></tr>
<tr><td>Target</td><td colspan="5">脚／関節識別子(F1～F3, J13,J23,J33: 省略不可)</td></tr>
<tr><td>ArgType1</td><td colspan="5">enum argument_typeT; 第一引数の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">enum argument_typeT; 第二引数の種類（無い時は ARG_NONE)</td></tr>
<tr><td>ArgType3</td><td colspan="5">enum argument_typeT; 第三引数の種類（無い時は ARG?NONE)</td></tr>
<tr><td>Value1</td><td colspan="5">第一引数の値（変数の時は添え字)</td></tr>
<tr><td>Value2</td><td colspan="5">第二引数の値（変数の時は添え字, 無いときは 0.0)</td></tr>
<tr><td>Value3</td><td colspan="5">第三引数の値（変数の時は添え字, 無いときは 0.0)</td></tr>
</table>

7-3-6 REL_COMPLIANCE_POSITION

REL_COMPLIANCE_POSITION ： 指先（など）にバネを設定する						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	REL_COMPLIANCE_POSITION (HS) (指/関節識別子) (R|A)(目標値１). . (R|A)(目標値３)					
種別	汎用					
引数	HS ：高速に移動することを指示する. 指識別子 ：Ｆ１，Ｆ２，Ｆ３ 関節識別子 ： Jn3 全指指定 ：ALL R|A ：直後の目標値が相対指令値(R)か絶対指令値(A)を指示する. A は省略できる. 目標値：定数あるいは変数. 定数の場合は，小数点および＋記号を省略可能. 相対座標.					
機能	目標位置からの距離に比例する力で引っ張るバネを指先（または指定の関節）に設定する. バネの剛性は，SET_POSITION_STIFFNESS コマンドで設定する. 単位は[N/mm]. Titan の場合は接触力を計測する手段が無いので難しいので，実装の予定は無い.					
注意	HiroshimaHand の場合， z 座標は無視される.					
例	REL_COMPLIANCE_POSITION F1 50.0 20.0 0.0 ：指 F1 の指先と相対座標(50.0, 20.0)の間にバネを設定する. REL_COMPLIANCE_POSITION J13 20.0 30.0 40.0 ：指 F1 の第 3 関節と相対座標(20.0, 30.0)の間にバネを設定する. 40.0 は無視される. REL_COMPLIANCE_POSITION J23 POS3 ：指 F2 の第 3 関節と相対座標(POS3.x, POS3.y)の間にバネを設定する. POS3.z は無視.					
See	COMPLIANCE_POSITION, NO_COMPLIANCE, SET_POSITION_STIFFNESS, PTP, REL_PTP, TORQUE					
メモリ格納形式	Cmd	REL_COMPLIANCE_POSITION / REL_COMPLIANCE_POSITION_HS				
	Target	脚／関節識別子(F1～F3, J13,J23,J33: 省略不可)				
	ArgType1	enum argument_typeT; 第一引数の種類				
	ArgType2	enum argument_typeT; 第二引数の種類（無い時は ARG_NONE)				
	ArgType3	enum argument_typeT; 第三引数の種類（無い時は ARG?NONE)				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字, 無いときは 0.0)				
	Value3	第三引数の値（変数の時は添え字, 無いときは 0.0)				

７-３-７　NO_COMPLIANCE

NO_COMPLIANCE　：　指定された脚／関節のコンプライアンス指定を解除						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	NO_COMPLIANCE （脚/関節識別子）					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４ 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３					
機能	指定された脚／関節のコンプライアンス指定を解除する(=100%)					
注意						
例	NO_COMPLIANCE F1					
See	PTP, PTP_ANGLE,PTP_TORQUE, NO_COMPLIANCE					
メモリ格納形式	Cmd	NO_COMPLIANCE				
	Target	関節識別子(J11～J43: 省略不可)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-4　WAIT 系コマンド

WAIT_INPOS/NEAR は，指定した足先位置や関節角度が目標位置に達したか，あるいは近傍に到着するまで待機する．トルク制御に関しては考慮に入れない．WAIT_INPOS/NEAR は，引数の種類や制御モード（位置制御，角度制御）に応じて引数の意味が変化するので要注意．WAIT_INPOS と WAIT_NEAR の機能上の違いは，WAIT_INPOS は引数を持たない（SET_JUST_POSITION で設定した値に従う）のに対して，WAIT_NEAR は引数として近傍位置／角度誤差を一時的に指定することができる（勿論，SET_NEAR_POSITION で設定値を変えることも可能）．

7-4-1　SLEEP

SLEEP　：　時間つぶし						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SLEEP （ミリ秒）					
種別	汎用					
引数	ミリ秒単位で指定する．汎用変数(VARnn)か即値のみ．					
機能	指定された時間だけ待機する．					
注意						
例	SLEEP　10　：10msec の時間つぶし SLEEP　VAR01　：VAR01 分の時間つぶし					
See						
メモリ格納形式	Cmd	SLEEP				
	Target	NO_TARGET				
	ArgType1	VAR，または，CONSTANT				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	汎用変数の添え字，または即値				
	Value2	0.0				
	Value3	0.0				

７-４-２　WAIT_INPOS

WAIT_INPOS　：　脚／関節が目標位置(角度)に到達するまで待機						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	WAIT_IN_POS　(脚/関節識別子/ALL)					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４ 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３ ALL　　：全ての位置/角度制御されている脚先座標あるいは関節					
機能	引数が, 1. 脚識別子の場合：指定した脚（の末端：脚先など）が位置制御モードの時は，指定した脚が目標位置±許容誤差[deg]に達するまで待つ．指定された脚の関節のうち一つでも関節角度制御モードの時は，指定した脚の関節角度制御された関節全てが目標角度±許容角度[deg]に達するまで待つ．位置制御，位置制御されていない場合は待機しない． 2. 関節識別子の場合：指定した関節が位置制御モードの時は，指定した関節が目標位置±許容誤差[mm]に達するまで待つ．指定した関節が関節角度制御モードの時は，指定した関節が目標角度±許容角度[deg]に達するまで待つ．指定した関節が位置制御，角度制御されていない場合は待機しない． 3. ALL の場合：全ての位置制御されている脚が目標位置±許容誤差[mm]，関節角度制御されている関節が目標角度±許容角度[deg]に達するまで待つ．位置制御，角度制御されている関節が存在しない場合は待機しない．					
注意						
例	WAIT_IN_POS　F1 ：脚 F1 の脚先座標が目標位置に達するまで待つ． WAIT_IN_POS　J12 ：脚 F1 の関節 J12 が目標角度に達するまで待つ．					
See	PTP, PTP_ANGLE, WAIT_NEAR, SET_JUST_POSITION					

メモリ格納形式		
	Cmd	WAIT_INPOS
	Target	脚／関節識別子(F1～F4，ALL, J11～J43: 省略不可)
	ArgType1	ARG_NONE
	ArgType2	ARG_NONE
	ArgType3	ARG_NONE
	Value1	0.0
	Value2	0.0
	Value3	0.0

7-4-3　WAIT_NEAR

<table>
<tr><th colspan="7">WAIT_NEAR　：　指定した脚／関節の目標位置近傍に到着するまで待機</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">WAIT_NEAR （脚/関節識別子/ALL）</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４
関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３
ALL　　：全ての位置/角度制御されている脚先座標あるいは関節</td></tr>
<tr><td>機能</td><td colspan="6">引数が,
1. 脚識別子の場合：指定した脚（の末端：脚先など）が位置制御モードの時は，指定した脚が目標位置±近傍誤差[deg]に達するまで待つ．指定された脚の関節のうち一つでも関節角度制御モードの時は，指定した脚の関節角度制御された関節全てが目標角度±近傍角度[deg]に達するまで待つ．位置制御，位置制御されていない場合は待機しない．
2. 関節識別子の場合：指定した関節が位置制御モードの時は，指定した関節が目標位置±近傍誤差[mm]に達するまで待つ．指定した関節が関節角度制御モードの時は，指定した関節が目標角度±近傍角度[deg]に達するまで待つ．指定した関節が位置制御，角度制御されていない場合は待機しない．
3. ALL の場合：全ての位置制御されている脚が目標位置±近傍誤差[mm]，関節角度制御されている関節が目標角度±近傍角度[deg]に達するまで待つ．位置制御，角度制御されている関節が存在しない場合は待機しない．</td></tr>
<tr><td>注意</td><td colspan="6">指定した脚あるいは関節が位置／関節角度制御モードにない場合はエラー(Invalid control mode)とする．トルク制御の場合には使用できない．
Ver.3.1 以降，許容誤差を引数に取らないように変更した．</td></tr>
<tr><td>例</td><td colspan="6">WAIT_NEAR　F1　10.0
：脚 F1 の脚先座標が目標位置の半径 10[mm]以内に達するまで待つ．
WAIT_NEAR　J12　5.0
：脚 F1 の関節 J12 が目標角度±5[deg]に達するまで待つ．
WAIT_NEAR_POS　J12　-VAR02
これは不可（たとえ VAR02 の中身が負数であっても）</td></tr>
<tr><td>See</td><td colspan="6">PTP, PTP_ANGLE,　WAIT_IN_POS, SET_NEAR_POSITION_POS</td></tr>
<tr><td rowspan="9">メモリ
格納形式</td><td>Cmd</td><td colspan="5">WAIT_NEAR</td></tr>
<tr><td>Target</td><td colspan="5">脚／関節識別子(F1～F4, ALL, J11～J43: 省略不可)</td></tr>
<tr><td>ArgType1</td><td colspan="5">ARG_NONE</td></tr>
<tr><td>ArgType2</td><td colspan="5">ARG_NONE</td></tr>
<tr><td>ArgType3</td><td colspan="5">ARG_NONE</td></tr>
<tr><td>Value1</td><td colspan="5">0.0</td></tr>
<tr><td>Value2</td><td colspan="5">0.0</td></tr>
<tr><td>Value3</td><td colspan="5">0.0</td></tr>
</table>

7-5　教示系コマンド

TEACH 系のコマンドは，関節毎の指定はできない．脚番号を指定し，構造体に一気に値をセットする．汎用変数を指定することもできず，TEACH_POS ならばポジション変数構造体，TEACH_ANGLE ならば関節角度変数構造体，TEACH_TORQUE ならば関節トルク変数構造体のみ指定可能．
（関節毎にセットしたり，違う型の変数にセットするような自由は無意味）

7-5-1　TEACH_POSITION

TEACH_POSITION　：　現在の脚先／関節の座標を教示						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	TEACH_POSITION （脚／関節識別子）（座標型変数構造体）					
種別	汎用					
引数	脚識別子：　F1,F2,F3,F4 関節識別子：J11, J12, J13, J21,...,J43					
機能	脚識別子により指定された脚の脚先の座標，または関節識別ににより指定された関節の座標を，座標型変数構造体の各メンバ(X,Y,Z)に代入する．					
注意	Ver.2.1 から名称を TEACH_POS -> TEACH_POSITION に変更． ただし互換性を残すために，TEACH_POS も使用可能					
例	TEACH_POSITION F1 POS01 ：脚 F1 の脚先座標を関節型変数 POS01 に代入する． TEACH_POSITION J23 POS02 ：脚 F2 の関節 2 の座標を関節型変数 POS02 に代入する．					
See	TEACH_ANGLE, TEACH_TORQUE, PTP					
メモリ格納形式	Cmd	TEACH_POSITION				
	Target	脚識別子(F1, F2, F3, F4) または関節識別子(J11, J12, J13, J21, ..., J43)．省略不可				
	ArgType1	POS				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	ポジション変数構造体の添え字				
	Value2	0.0				
	Value3	0.0				

7-5-2　TEACH_ANGLE

TEACH_ANGLE　：　現在の関節角度を教示						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	TEACH_ANGLE （脚／関節識別子）（関節型変数構造体／汎用変数または関節型変数メンバ）					
種別	汎用					
引数	脚識別子：　F1,F2,F3,F4 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３					
機能	脚識別子により指定された脚の関節角度を関節型変数構造体の各メンバ(J1,J2,J3)に代入する．関節識別子が指定された場合は，指定された関節の関節角度を変数（関節型変数のメンバ含む）に代入する．					
注意	ALL は指定できない					
例	TEACH_ANGLE J11 AGL01 ：脚 F1 の各関節角度を関節型変数 AGL01 に代入する．					
See	TEACH_POSITION, TEACH_TORQUE, PTP_ANGLE					
メモリ格納形式	Cmd	TEACH_ANGLE				
	Target	脚識別子(F1, F2, F3, F4)／関節識別子(Jn1, Jn2, Jn3...): 省略不可				
	ArgType1	変数の型				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	変数の添え字				
	Value2	0.0				
	Value3	0.0				

7-5-3　TEACH_TORQUE

TEACH_TORQUE　：　現在の関節トルクを教示						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	TEACH_TORQUE （脚／関節識別子）（関節型変数構造体／汎用変数または関節型変数メンバ）					
種別	汎用					
引数	脚識別子：　F1,F2,F3,F4 関節識別子　：Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３					
機能	脚識別子により指定された脚の関節トルクを関節型変数構造体の各メンバ(J1,J2,J3)に代入する．関節識別子が指定された場合は，指定された関節の関節トルクを変数（関節型変数のメンバ含む）に代入する．					
注意	ALL は指定できない					
例	TEACH_TORQUE J11 TRQ01 ：F1 の各関節トルクを関節型変数 TRQ01 に代入する．					
See	TEACH_POSITION, TEACH_ANGLE, TORQUE					
メモリ格納形式	Cmd	TEACH_TORQUE				
	Target	脚識別子(F1, F2, F3, F4)／関節識別子(Jn1, Jn2, Jn3...): 省略不可				
	ArgType1	変数の型				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	変数の添え字				
	Value2	0.0				
	Value3	0.0				

7-5-4 TEACH_FORCE

TEACH_FORCE　：　現在の接触力ベクトルを教示						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	TEACH_FORCE （指／関節識別子）（ポジション変数構造体）					
種別	汎用					
引数	指識別子：　F1,F2,F3(指先) 関節識別子：J12, J22, J32(第２関節)					
機能	指識別子により指定された指の指先の接触力ベクトル，または関節識別ににより指定された第２関節に働く接触力ベクトルを，ポジション変数構造体の各メンバ(X,Y,Z)に代入する． 関節トルクセンサで検出した関節トルクに，$J^{T\#}$（第２関節は$(J_2^T)^{-1}$）を掛けて接触力ベクトルを得る．					
注意	接触力の単位は[N]（のはず） 接触力ベクトルの成分のうち，ｚ成分は常に 0.0．ただし，擬似逆行列（$J^{T\#}$）が取れない場合は接触力は(0.0, 0.0, -1.0)となる．					
例	TEACH_FORCE F1 POS01 ：指 F1 の指先に働く接触力ベクトルをポジション変数 POS01 に代入する． TEACH_FORCE J23 POS02 ：指 F2 の関節 2 に働く接触力ベクトルをポジション変数 POS02 に代入する．					
See	TEACH_POSITION, TEACH_ANGLE, TEACH_TORQUE, FORCE					
メモリ格納形式	Cmd	TEACH_FORCE				
	Target	脚識別子(F1, F2, F3) または関節識別子(J12, J22, J32)．省略不可				
	ArgType1	POS				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	ポジション変数構造体の添え字				
	Value2	0.0				
	Value3	0.0				

７-６ 制御構文

制御構文として，以下のコマンドが用意されている．

(1) ブロックＩＦ文（IF_THEN-ELSE-ENDIF）

条件式が成り立つならば，ステートメントブロック１（IF-THEN と ELSE で囲まれたステートメントブロック）を，成り立たないならばステートメントブロック２（ELSE と ENDIF に囲まれたステートメントブロック）を実行する．

(2) ＷＨＩＬＥ文（WHILE-WEND）

条件式が成り立つ間は WHILE と WEND で囲まれたステートメントブロックを実行する．条件判断はステートメントブロック処理前に行われる．

(3) ＲＥＰＥＡＴ文（REPEAT-UNTIL）

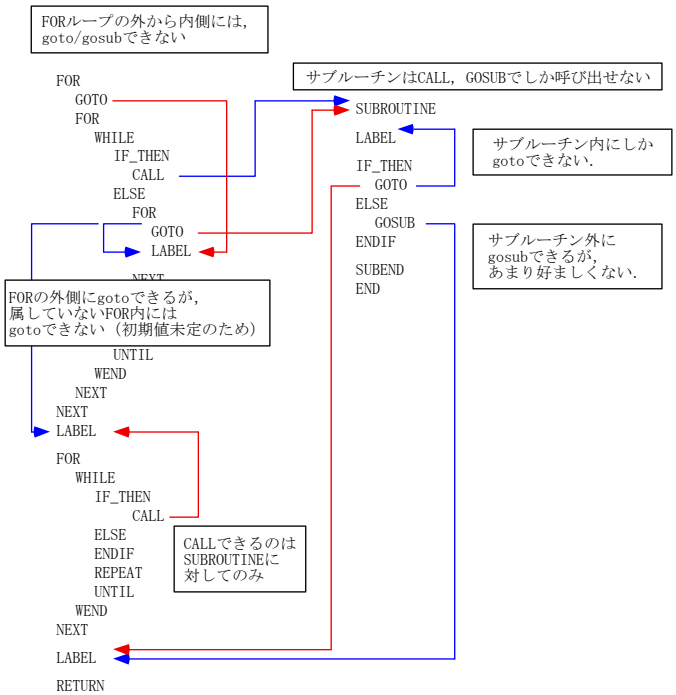
条件式が成り立たない間，REPEAT と UNTIL に囲まれたステートメントブロックを実行する．条件判断はステートメントブロック処理後に行われる．

(4) ＦＯＲ文（FOR-NEXT）

汎用変数に初期値を代入し，その値が終わり値になるまで FOR と NEXT で囲まれたステートメントブロックを実行する．ステートメントブロック処理後に汎用変数は＋１カウントアップする．

(5) サブルーチン（SUBROUTINE-SUBEND/CALL）

SUBROUTINE と SUBEND で囲まれたステートメントブロックに対して，外部から goto 系コマンド（GOTO, IF_GOTO）でジャンプしてくることはできない．まだ，外部に goto 系コマンドで抜け出すこともできない．呼び出すには，CALL コマンドか，gosub 系コマンド（GOSUB, IF_GOSUB）を用いる．サブルーチン内のラベルに対しても同様である．



<table>
<tr><th colspan="6">IF_THEN_ELSE_ENDIF/ELSEIF　：　ブロック IF 文</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="3">IF （条件式） THEN
　　（ステートメント・ブロック）
ENDIF

IF （条件式） THEN
　　（ステートメント・ブロック１）
ELSE
　　（ステートメント・ブロック２）
ENDIF</td><td colspan="3">IF（条件式）THEN
　　（ステートメント・ブロック１）
ELSEIF （条件式） THEN
　　（ステートメント・ブロック２）
ELSEIF （条件式） THEN
　　（ステートメント・ブロック３）
ELSE
　　（ステートメント・ブロック４）
ENDIF</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">条件式：　変数あるいは定数間の関係演算式，あるいはイベントフラグ</td></tr>
<tr><td>機能</td><td colspan="6">（条件式）が真ならばステートメント・ブロック１，偽ならばステートメント・ブロック２が実行される．条件式については，詳しくは 関係演算子６－１節を参照．ELSEIF で条件式を連ねていくことも可能．</td></tr>
<tr><td>注意</td><td colspan="6">ステートメント・ブロック内に他の制御構文をネストすることは可能
ELSE 行，ENDIF 行はシングルステートメント．</td></tr>
<tr><td>例</td><td colspan="3">IF　POS01.X == POS02.X THEN
　　GOSUB L2:
　　VAR10 -= VAR20
ENDIF

IF　POS01.X == POS02.X THEN
　　GOSUB L1:
　　VAR10 += VAR20
ELSE
　　GOSUB L2:
　　VAR10 -= VAR20
ENDIF</td><td colspan="3">IF　VAR_KEY == CHAR_A THEN
　　MESSAGE 処理１
ELSEIF VAR_KEY == CHAR_B THEN
　　MESSAGE 処理２
ELSEIF VAR_KEY == CHAR_C THEN
　　MESSAGE 処理３
ELSE
　　GOSUB L4:
ENDIF</td></tr>
<tr><td>See</td><td colspan="6">GOTO，ラベル，IF_GOTO，IF_GOSUB，GOSUB，イベントフラグ</td></tr>
</table>

メモリ格納形式(IF)	Cmd	IF_THEN_？？(EQ, NE, GT, LT, GE, LE, EVENT)
	Target	NO_TARGET
	ArgType1	条件式左辺の種類：EVENT_FLAG の時は EVENT の種類
	ArgType2	条件式右辺の種類：EVENT_FLAG の時は ARG_NONE
	ArgType3	LABEL
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAG の時は，肯定ならば 1.0，否定ならば -1.0
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAG の時は 0.0
	Value3	対応する ELSE/ELSEIF/ENDIF 行のプログラムステップ番号
メモリ格納形式(ELSEIF)	Cmd	ELSEIF_？？(EQ, NE, GT, LT, GE, LE, EVENT)
	Target	NO_TARGET
	ArgType1	条件式左辺の種類：EVENT_FLAG の時は EVENT の種類
	ArgType2	条件式右辺の種類：EVENT_FLAG の時は ARG_NONE
	ArgType3	LABEL
	ArgType4	CONSTANT
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAG の時は，肯定ならば 1.0，否定ならば -1.0
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAG の時は 0.0
	Value3	対応する ELSE/ELSEIF/ENDIF 行のプログラムステップ番号
	Value4	対応する IF_THEN_??あるいは ELSEIF_??によって書き換えられる． (1) IF_THEN_??：条件式が真の場合は 1，偽の場合は -1 (2) ELSEIF_??：条件式が真の場合，あるいは value4 が 1 の場合は 1，偽の場合は-1 既に他のステートメント・ブロックを実行したのならば，それ以外のブロックは素通りしなくてはならないためである．Value4 が-1 の時は条件判断を行う（その結果を次の ELSEIF あるいは ELSE の Value4 に反映する）．1 の場合は無条件に Value3 のステップ番号にジャンプすると同時に，もし ELSEIF あるいは ELSE が続くならば Value4 に反映する．
メモリ格納形式(ELSE)	Cmd	ELSE
	Target	NO_TARGET
	ArgType1	LABEL
	ArgType2	CONSTANT
	ArgType3	ARG_NONE
	Value1	対応する ENDIF 行のプログラムステップ番号
	Value(2～3)	0.0
	Value4	対応する IF_THEN_??あるいは ELSEIF_??によって書き換えられる． (1) IF_THEN_??：条件式が真の場合は 1，偽の場合は -1 (2) ELSEIF_??：条件式が真の場合，あるいは value4 が 1 の場合は 1，偽の場合は-1 既に他のステートメント・ブロックを実行したのならば，それ以外のブロックは素通りしなくてはならないためである．Value4 が-1 の時はステートメント・ブロックを実行する．1 の場合は何も行わずに ENDIF へジャンプする．
メモリ格納形式(ENDIF)	Cmd	ENDIF
	Target	NO_TARGET
	ArgType(1～3)	ARG_NONE
	Value(1～3)	0.0

7-6-2 WHILE_WEND

WHILE_WEND : while 文						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	WHILE （条件式） （ステートメント・ブロック） WEND					
種別	汎用					
引数	条件式： 変数あるいは定数間の関係演算式，あるいはイベントフラグ					
機能	（条件式）が真の間，ステートメントブロックが実行され，条件が偽になったら，WEND 行の次の行に処理を移す. 条件式については，詳しくは 関係演算子６－１節を参照					
注意	ステートメント・ブロック内に他の制御構文をネストすることは可能 WEND 行はシングルステートメント.					
例	WHILE POS01.X <= POS02.X GOSUB L1: VAR10 += VAR20 WEND					
See	GOTO，ラベル, IF_GOTO, IF_GOSUB, GOSUB，イベントフラグ					
メモリ格納形式 (WHILE)	Cmd	WHILE_？？(EQ, NE, GT, LT, GE, LE, EVENT)				
	Target	NO_TARGET				
	ArgType1	条件式左辺の種類：EVENT_FLAG の時は EVENT の種類				
	ArgType2	条件式右辺の種類：EVENT_FLAG の時は ARG_NONE				
	ArgType3	LABEL				
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAG の時は，肯定ならば 1.0，否定ならば -1.0				
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAG の時は 0.0				
	Value3	対応する WEND 行の次の行のステップ番号				
メモリ格納形式 (WEND)	Cmd	WEND				
	Target	NO_TARGET				
	ArgType1	LABEL				
	ArgType (2～3)	ARG_NONE				
	Value1	対応する WHILE 行のプログラムステップ番号				
	Value (2～3)	0.0				

REPEAT_UNTIL　：　repeat-until 文						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	REPEAT （ステートメント・ブロック） UNTIL （条件式）					
種別	汎用					
引数	条件式：　変数あるいは定数間の関係演算式，あるいはイベントフラグ					
機能	（条件式）が真になるまで，ステートメントブロックを実行する．条件が真になったら，UNTIL 行の次の行に処理を移す． 条件式については，詳しくは 関係演算子６－１節を参照					
注意	ステートメント・ブロック内に他の制御構文をネストすることは可能 REPEAT 行はシングルステートメント． Ｃ言語の do-while 文ではなく，Pascal の repeat until 文．until 行の条件式の論理が do-while とは逆である．					
例	REPEAT GOSUB L1: VAR10 += VAR20 UNTIL　POS01.X <= POS02.X					
See	GOTO，ラベル, IF_GOTO, IF_GOSUB, GOSUB，イベントフラグ					
メモリ格納形式(REPEAT)	Cmd	REPEAT				
	Target	NO_TARGET				
	ArgType (1～3)	ARG_NONE				
	Value (1～3)	0.0				
メモリ格納形式(UNTIL)	Cmd	UNTIL_？？(EQ, NE, GT, LT, GE, LE, EVENT)				
	Target	NO_TARGET				
	ArgType1	条件式左辺の種類：EVENT_FLAG の時は EVENT の種類				
	ArgType2	条件式右辺の種類：EVENT_FLAG の時は ARG_NONE				
	ArgType3	LABEL				
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAG の時は，肯定ならば 1.0，否定ならば -1.0				
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAG の時は 0.0				
	Value3	対応する REPEAT のステップ番号				

7-6-4 FOR_NEXT

FOR_NEXT　：　for-next 文

対応	TitanVIII	○	HiroshimaHand	○	汎用	○

書式

FOR （汎用型変数） =（初期値） TO （終端値） STEP （刻み値）
　　（ステートメント・ブロック）
NEXT （汎用型変数）

種別

汎用

引数

初期値：　即値，汎用型変数，あるいは構造体型変数のメンバー
終端値：　即値，汎用型変数，あるいは構造体型変数のメンバー
刻み値：　即値，汎用型変数，あるいは構造体型変数のメンバー（省略時は１）

汎用型変数：NEXT の汎用型変数は省略可能．その時は，FOR-NEXT の厳密な対応はチェックしない．

機能

(1)（汎用型変数）に初期値を代入する．
　＜（刻み値）が正の場合，および省略時＞
　（汎用型変数）＜＝（終端値）の間は，ステートメント･ブロックを実行する．
　＜（刻み値）が負の場合＞
　（汎用型変数）＞＝（終端値）の間は，ステートメント･ブロックを実行する．
(2) ステートメント･ブロック実行後，（汎用型変数）に（刻み値）を加算する．（刻み値）省略時は 1.0 が加算される．
(3) 条件が満たされたら NEXT 行の次の行に処理を移す．

注意

FOR の（汎用型変数）と NEXT の（汎用型変数）は同一でなくてはならない．
ステートメント・ブロック内に他の制御構文をネストすることは可能
ステートメント・ブロック内でラベルの使用は可である．ステートメント・ブロック内外への出入りは自由とする．ただし，まだ実行されていない FOR-NEXT に外部から入った場合，汎用型変数を増分する際に実行時エラーとなる．

例

```
FOR VAR10 = POS1.X TO 10.0
    GOSUB L1:
    VAR10 += VAR20
NEXT VAR10
```

See

GOTO，ラベル, IF_GOTO, IF_GOSUB, GOSUB，イベントフラグ

メモリ格納形式 (FOR)

Cmd	FOR
Target	NO_TARGET
ArgType1	VAR
ArgType2	初期値の種類
ArgType3	終端値の種類
ArgType4	LABEL
ArgType5	刻み地の種類（省略時は CONSTANT)
Value1	汎用変数の添え字
Value2	初期値の値（変数の場合は添え字)
Value3	終端値の値（変数の場合は添え字)
Value4	NEXT 行の次の行のプログラムステップ番号
Value5	刻み値の値（変数の場合は添え字），省略時は 1.0

メモリ格納形式 (NEXT)

Cmd	NEXT
Target	NO_TARGET
ArgType1	VAR
ArgType2	LABEL
ArgType3	ARG_NONE
Value1	汎用型変数の添え字
Value2	FOR 行のプログラムステップ番号
Value3	0.0

7-6-5 SUBROUTINE-SUBEND

SUBROUTINE_SUBEND ： サブルーチン						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SUBROUTINE ラベル名： （ステートメント・ブロック） SUBEND					
種別	汎用					
引数	ラベル名： GO/GOSUB などで用いられるラベルと同じ					
機能	CALL あるいは GOSUB によってコールされる. SUBEND で（明示的に RETURN を記述しないでも）RETURN する. 文法チェックを厳しくする効果がある.					
注意	ステートメント・ブロック内に他の制御構文をネストすることは可能 SUBROUTINE 外から中へ，あるいは中から外へ GOTO 系コマンドによって出入りすることはできないように制限される．これはカプセル化を強めるための制約である.					

例	CALL KEY_CHECKER:
	
	SUBROUTINE KEY_CHECKER:
	IF KB_HIT THEN
	KEY_IN
	ELSE
	VAR_KEY = -1.0
	ENDIF
	SUBEND					

See	GOSUB, ラベル, IF_GOSUB, CALL					
メモリ格納形式 (SUBROUTINE)	Cmd	SUBROUTINE				
	Target	NO_TARGET				
	ArgType (1～3)	ARG_NONE				
	Value (1～3)	0.0				
メモリ格納形式 (SUBEND)	Cmd	SUBEND				
	Target	NO_TARGET				
	ArgType (1～3)	ARG_NONE				
	Value (1～3)	0.0				

7-6-6 CALL

CALL　：　サブルーチン・コール						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	CALL ラベル名：					
種別	汎用					
引数	ラベル名： GO/GOSUB などで用いられるラベルと同じ					
機能	SUBROUTINE を呼び出す.					
注意	SUBROUTINE 以外のラベルをコールすることはできない.					

例	CALL KEY_CHECKER:
	
	SUBROUTINE KEY_CHECKER:
	　IF KB_HIT THEN
	　　KEY_IN
	　ELSE
	　　VAR_KEY = -1.0
	　ENDIF
	SUBEND					

See	GOSUB, ラベル, IF_GOSUB, SUBROUTINE					
メモリ格納形式	Cmd	CALL				
	Target	NO_TARGET				
	ArgType1	LABEL				
	ArgType (2～3)	ARG_NONE				
	Value1	SUBROUTINE のステップ番号				
	Value (2～3)	0.0				

7-6-7 IF_GOTO

IF_GOTO　：　条件分岐(GOTO)						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	IF （条件式） GOTO （ラベル名:）					
種別	汎用					
引数	条件式：　変数あるいは定数間の関係演算式，あるいはイベントフラグ ラベル：　省略可能．省略された場合はLOOPと解釈される．					
機能	（条件式）が真ならば（ラベル）へジャンプする． 条件式については，詳しくは 関係演算子６－１節を参照					
注意						
例	IF　POS01.X == POS02.X GOTO L10: ：POS01.XとPOS02.Xが等しいならばL10へジャンプする． IF　VAR01 < 10.0 GOTO END_ROUTINE: ：汎用変数VAR01の内容が10.0よりも小さいならばEND_ROUTINEへジャンプする．					
See	GOTO，ラベル，IF_GOSUB，GOSUB，イベントフラグ					
メモリ格納形式	Cmd	IF_GOTO_？？(EQ, NE, GT, LT, GE, LE, EVENT)				
	Target	NO_TARGET				
	ArgType1	条件式左辺の種類：EVENT_FLAGの時はEVENTの種類				
	ArgType2	条件式右辺の種類：EVENT_FLAGの時はARG_NONE				
	ArgType3	LABEL				
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAGの時は，肯定ならば1.0，否定ならば -1.0				
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAGの時は0.0				
	Value3	プログラムステップ番号				

7-6-8　IF_GOSUB

IF_GOSUB　：　条件分岐(GOSUB)						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	IF （条件式） GOSUB （ラベル名:）					
種別	汎用					
引数	条件式：　変数あるいは定数間の関係演算式，あるいはイベントフラグ ラベル：　省略不可能．LOOP 使用禁止．					
機能	（条件式）が真ならば（ラベル）へジャンプする．RETURN コマンドで次のプログラムステップに戻る．条件式については，詳しくは 関係演算子６－１節を参照．					
注意						
例	IF　POS01.X == POS02.X GOSUB L10: ：POS01.X と POS02.X が等しいならば L10 へジャンプする． IF　VAR01 < 10.0 GOSUB END_ROUTINE: ：汎用変数 VAR01 の内容が 10.0 よりも小さいならば END_ROUTINE へジャンプする．					
See	GOTO，ラベル，RETURN，GOSUB，IF_GOTO，イベントフラグ					
メモリ格納形式	Cmd	IF_GOSUB_？？(EQ, NE, GT, LT, GE, LE, EVENT)				
	Target	NO_TARGET				
	ArgType1	条件式左辺の種類：EVENT_FLAG の時は EVENT の種類				
	ArgType2	条件式右辺の種類：EVENT_FLAG の時は ARG_NONE				
	ArgType3	LABEL				
	Value1	条件式左辺の値（変数の場合は添え字） EVENT_FLAG の時は，肯定ならば 1.0，否定ならば -1.0				
	Value2	条件式右辺の値（変数の場合は添え字）：EVENT_FLAG の時は 0.0				
	Value3	プログラムステップ番号				

7-7　ジャンプ系コマンド

特別なジャンプ先として，ＬＯＯＰ（繰り返し使用可能）がある．

7-7-1　GOTO

GOTO　：　指定したラベル行へジャンプ						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	GOTO （ラベル名：）					
種別	汎用					
引数	ラベル名：省略可能．省略した場合はLOOPと解釈される．					
機能	無条件に対応するラベルの存在するプログラムステップにジャンプする．ラベルが省略された場合は LOOP と解釈される．その場合，最も近い上流に存在する LOOP:ラベルコマンドの存在する行へジャンプする．ごく小さな繰り返しルーチンでの使用に適している．反面，大きなルーチンの繰り返しに用いると，入れ子になった時に，思わぬトラブルを引き起こすことになる．					
注意						
例	GOTO　STARTING_WALK: ：ラベル"STARTING_WALK:"にジャンプする． GOTO　LOOP: ：もちろん"LOOP"を明示しても問題はない．					
See	ラベル, IF_GOTO, GOSUB, IF_GOSUB					
メモリ格納形式	Cmd	GOTO				
	Target	NO_TARGET				
	ArgType1	LABEL				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	プログラムステップ番号				
	Value2	0.0				
	Value3	0.0				

7-7-2　GOSUB

GOSUB　:　指定されたラベル行へサブルーチンコール						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	GOSUB （ラベル名：）					
種別	汎用					
引数	ラベル：省略不可能．LOOP: の使用は禁止する．					
機能	無条件に対応するラベルの存在するプログラムステップにジャンプする． RETURN コマンドで次のステップに戻る．多重コールも可能とする．					
注意						
例	GOSUB　STARTING_WALK: ：ラベル"STARTING_WALK:"にジャンプする． RETURN があったら次のプログラムステップに戻る．					
See	**ラベル, IF_GOSUB, RETURN, IF_GOTO**					
メモリ格納形式	Cmd	GOSUB				
	Target	NO_TARGET				
	ArgType1	LABEL				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	プログラムステップ番号				
	Value2	0.0				
	Value3	0.0				

7-7-3　RETURN

RETURN　:　サブルーチンコール元(GOSUB)に戻る						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	RETURN					
種別	汎用					
引数	なし					
機能	GOSUB, IF_GOSUB によりジャンプしてきたサブルーチンから，一つ前のルーチン（GOSUB,IF_GOSUB コマンドの次のプログラムステップ）に戻る．GOSUB/IF_GOSUB との対応が取れていない場合は，エラー(Invalid Return command)を発生する．					
注意						
例	GOSUB　STARTING_WALK: VAR01 = VAR02　←―――――＋ ｜ STARTING_WALK :　｜ ......　｜ RETURN　――――――――＋					
See	**ラベル, IF_GOSUB, GOSUB**					
メモリ格納形式	Cmd	RETURN				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

７-８　パラメータ調整系コマンド[10]

各速度やトルクに関するパラメータは，TIP 実行時には初期値に設定されているパラメータを一時的に書き換えたり，あるいは一時的に大きく(FAST,FIRMLY)／小さく(SLOW/SOFT)したり，戻したり(NORMAL)する．ここで書き換えたパラメータは，AUTO モードを抜けると初期値（MENU にて設定した値）に書き戻される．

７-８-１ 位置制御系パラメータ

７-８-１-１　SET_A_MAX

SET_A_MAX　：　脚先位置制御時の最大加速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_A_MAX （脚識別子） 指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[mm/sec^2]．					
機能	指定された脚の加速度制限値(A_MAX)を一時的に変更する．					
注意	指令値が負，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする．					
例	SET_A_MAX　F1　50.0 ：脚 F1 の加速度制限値を 50.0[mm/sec^2]に設定する． SET_A_MAX　ALL VAR03 ：全ての脚の加速度制限値を VAR03 の値に設定する．					
See	SET_V_MAX, SET_A_MAX_SLOW, SET_A_MAX_FAST, SET_A_MAX_NORMAL					
メモリ格納形式	Cmd	SET_A_MAX				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

[10] Ver.2.0 からコマンド名の頭に SET_が付加されたので注意.それ以前のバージョンの TIP プログラムは頭に SET_を追加する必要がある．

7-8-1-2　SET_V_MAX

SET_V_MAX　：　脚先位置制御時の最大速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_V_MAX　(脚識別子)　指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[mm/sec]．					
機能	指定された脚の速度制限値(V_MAX)を一時的に変更する．					
注意	指令値が負，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする．					
例	SET_V_MAX　F1　50.0 ：脚 F1 の速度制限値を 50.0[mm/sec]に設定する． SET_V_MAX　ALL VAR03 ：全ての脚の速度制限値を VAR03 の値に設定する．					
See	SET_A_MAX, SET_V_MAX_SLOW, SET_V_MAX_FAST, SET_V_MAX_NORMAL					
メモリ格納形式	Cmd	SET_V_MAX				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-1-3 SET_A_MAX_RATE

SET_A_MAX_RATE　：　最大加速度(A_MAX)を比で変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_A_MAX_RATE　(脚識別子)　指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[%]．					
機能	指定された脚の加速度制限値(A_MAX)を一時的に標準値の(指令値)倍する．					
注意	指令値≦０，指令値＞100，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする．					
例	SET_A_MAX_RATE　F1　50.0 ：脚 F1 の加速度制限値を標準の 50.0[%]に設定する． SET_A_MAX_RATE　ALL VAR03 ：全ての脚の加速度制限値を VAR03[%]に設定する．					
See	SET_A_MAX, SET_A_MAX_FAST / SLOW / NORMAL					
メモリ格納形式	Cmd	SET_A_MAX_RATE				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-1-4　SET_V_MAX_RATE

SET_V_MAX_RATE　：　最大速度(V_MAX)を比で変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_V_MAX_RATE （脚識別子） 指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[%]．					
機能	指定された脚の速度制限値(V_MAX)を一時的に(指令値) 倍に変更する．					
注意	指令値≦０，指令値＞100，　あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする．					
例	SET_V_MAX_RATE　F1　50.0 ：脚 F1 の速度制限値を 50.0[%]に設定する． SET_V_MAX_RATE　ALL VAR03 ：全ての脚の速度制限値を VAR03 倍に設定する．					
See	SET_V_MAX, SET_V_MAX_FAST / SLOW / NORMAL					
メモリ格納形式	Cmd	SET_V_MAX_RATE				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-1-5　SET_A_MAX_SLOW／FAST／NORMAL

SET_A_MAX_SLOW／FAXT／NORMAL　：　最大加速度を変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_A_MAX_SLOW　(脚識別子) SET_A_MAX_FAST　(脚識別子) SET_A_MAX_NORMAL　(脚識別子)					
種別	汎用					
引数	脚識別子　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ					
機能	指定された脚の加速度制限値(A_MAX)を一時的に変更する. SET_A_MAX_SLOW で, A_MAX の TIP_A_MAX_SLOW_RATE 倍(50%), SET_A_MAX_FAST で, A_MAX の TIP_A_MAX_FAST_RATE 倍(150%), SET_A_MAX_NORMAL で, A_MAX に戻す(100%).					
注意						
例	SET_A_MAX_FAST　F1 ：脚 F1 の加速度制限値を A_MAX の TIP_A_MAX_FAST 倍に設定する. SET_A_MAX _NORMAL ALL ：全ての脚の加速度制限値を A_MAX に戻す.					
See	SET_A _MAX, SET_A_MAX_RATE					

メモリ格納形式		
	Cmd	SET_A_MAX_SLOW, A_MAX_FAST, A_MAX_NORMAL
	Target	脚識別子(F1,F2,F3,F4,ALL)
	ArgType1	ARG_NONE
	ArgType2	ARG_NONE
	ArgType3	ARG_NONE
	Value1	0.0
	Value2	0.0
	Value3	0.0

7-8-1-6　SET_V_MAX_SLOW／FAST／NORMAL

SET_V_MAX_SLOW／FAST／NORMAL　：　最大速度を変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_V_MAX_SLOW　(脚識別子) SET_V_MAX_FAST　(脚識別子) SET_V_MAX_NORMAL　(脚識別子)					
種別	汎用					
引数	脚識別子　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ					
機能	指定された脚の速度制限値(V_MAX)を一時的に変更する. SET_V_MAX_SLOW で, V_MAX の TIP_V_MAX_SLOW_RATE 倍(50%), SET_V_MAX_FAST で, V_MAX の TIP_V_MAX_FAST_RATE 倍(150%), SET_V_MAX_NORMAL で, V_MAX に戻す(100%)					
注意						
例	SET_V_MAX_FAST　F1 ：脚 F1 の速度制限値を V_MAX の TIP_V_MAX_FAST 倍に設定する. SET_V_MAX _NORMAL ALL ：全ての脚の加速度制限値を V_MAX に戻す.					
See	SET_V_MAX, SET_V_MAX_RATE					
メモリ格納形式	Cmd	SET_V_MAX_SLOW, V_MAX_FAST, V_MAX_NORMAL				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-1-7 SET_JUST_POSITION

SET_JUST_POSITION　：　脚先位置制御時の許容誤差の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_JUST_POSITION （脚識別子） 指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[mm]．					
機能	指定された脚の位置制御許容誤差(INPOS)を一時的に変更する．					
注意	指令値が負，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする． Ver.3.1 から名前が SET_INPOS より変更になった．					
例	SET_JUST_POSITION　F1　5.0 ：脚 F1 の位置制御許容誤差を 5.0[mm]に設定する． SET_JUST_POSITION　ALL VAR03 ：全ての脚の位置制御許容誤差を VAR03 の値に設定する．					
See	SET_NEAR_POSITION, PTP, REL_PTP, WAIT_INPOS					
メモリ格納形式	Cmd	SET_JUST_POSITION				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-1-8 SET_NEAR_POSITION

SET_NEAR_POSITION　：　脚先位置制御時の近傍到着許容誤差の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_NEAR_POSITION （脚識別子） 指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 指令値　　：定数あるいは汎用変数．正の数字．単位は[mm]．					
機能	指定された脚の目標位置近傍許容誤差(NEAR)を一時的に変更する．					
注意	指令値が負，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする． Ver.3.1 から名前が SET_NEAR より変更になった．					
例	SET_NEAR_POSITION　F1　5.0 ：脚 F1 の目標位置近傍許容誤差を 5.0[mm]に設定する． SET_NEAR_POSITION　ALL VAR03 ：全ての脚の目標位置近傍許容誤差を VAR03 の値に設定する．					
See	SET_JUST_POSITION, PTP, REL_PTP, COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION, WAIT_NEAR					
メモリ格納形式	Cmd	SET_NEAR_POSITION				
	Target	脚識別子(F1,F2,F3,F4,ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-2-1 SET_OMEGA_MAX

SET_OMEGA_MAX : 関節角度制御時の最大関節角速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_MAX (関節識別子) 指令値					
種別	汎用					
引数	関節識別子 : J11, J12, J13, ..., J43, ALL 指令値 : 定数あるいは汎用変数. 正の数字. 単位は[deg/sec].					
機能	指定された関節の角速度制限値(OMEGA_MAX)を一時的に変更する.					
注意	指令値が負, あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_OMEGA_MAX J11 50.0 : 関節 J11 の角速度制限値を 50.0[deg/sec]に設定する. SET_OMEGA_MAX J33 VAR03 : 関節 J33 の角速度制限値を VAR03 の値に設定する.					
See	SET_OMEGA_DOT_MAX, SET_OMEGA_MAX_SLOW, SET_OMEGA_MAX_FAST, SET_OMEGA_MAX_NORMAL					
メモリ格納形式	Cmd	SET_OMEGA_MAX				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値 (変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-2-2　SET_OMEGA_DOT_MAX

SET_OMEGA_DOT_MAX　：　関節角度制御時の最大関節角加速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_DOT_MAX　(関節識別子)　指令値					
種別	汎用					
引数	関節識別子　　：J11, J12, J13, ..., J43, ALL 指令値　　：定数あるいは汎用変数. 正の数字. 単位は[deg/sec^2].					
機能	指定された関節の角加速度制限値(OMEGA_DOT_MAX)を一時的に変更する.					
注意	指令値が負, あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_OMEGA_DOT_MAX　J11　50.0 ：関節 J11 の角加速度制限値を 50.0[deg/sec^2]に設定する. SET_OMEGA_DOT_MAX　J33 VAR03 ：関節 J33 の角加速度制限値を VAR03 の値に設定する.					
See	SET_OMEGA_MAX, SET_OMEGA_DOT_MAX_SLOW, SET_OMEGA_DOT_MAX_FAST, SET_OMEGA_DOT_MAX_NORMAL					
メモリ格納形式	Cmd	SET_OMEGA_DOT_MAX				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値 (変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-2-3　SET_OMEGA_MAX_RATE

SET_OMEGA_MAX_RATE　：　最大角速度を比で変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_MAX_RATE　(関節識別子)　指令値					
種別	汎用					
引数	関節識別子　　：J11, J12, J13, ..., J43, ALL 指令値　　：定数あるいは汎用変数. 正の数字. 単位は[%].					
機能	指定された関節の角速度制限値(OMEGA_MAX)を一時的に(指令値)倍する.					
注意	指令値≦０, 指令値＞100,　あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_OMEGA_MAX_RATE　J11　50.0 ：関節 J11 の角速度制限値を 50.0[%]に設定する. SET_OMEGA_MAX_RATE　J33 VAR03 ：関節 J33 の角速度制限値を VAR03 倍に設定する.					
See	SET_OMEGA_ MAX,　SET_OMEGA_MAX_SLOW, SET_OMEGA_MAX_FAST,　SET_OMEGA_MAX_NORMAL					
メモリ格納形式	Cmd	SET_OMEGA_MAX_RATE				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値 (変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-2-4　SET_OMEGA_DOT_MAX_RATE

SET_OMEGA_DOT_MAX_RATE　：　最大角加速度を比で変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_DOT_MAX_RATE （関節識別子） 指令値					
種別	汎用					
引数	関節識別子　　：J11, J12, J13, ..., J43, ALL 指令値　　：定数あるいは汎用変数. 正の数字. 単位は[%].					
機能	指定された関節の角加速度制限値(OMEGA_DOT_MAX)を一時的に(指令値)倍する.					
注意	指令値≦０, 指令値＞100, あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_OMEGA_DOT_MAX_RATE J11 50.0 ：関節 J11 の角加速度制限値を 50.0[deg/sec^2]に設定する. SET_OMEGA_DOT_MAX_RATE J33 VAR03 ：関節 J33 の角加速度制限値を VAR03 の値に設定する.					
See	SET_OMEGA_DOT_MAX, SET_OMEGA_DOT_MAX_SLOW, SET_OMEGA_DOT_MAX_FAST, SET_OMEGA_DOT_MAX_NORMAL					

メモリ格納形式		
	Cmd	SET_OMEGA_DOT_MAX_RATE
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)
	ArgType1	指令値の種類
	ArgType2	ARG_NONE
	ArgType3	ARG_NONE
	Value1	指令値の値（変数の場合は添え字)
	Value2	0.0
	Value3	0.0

7-8-2-5　SET_OMEGA_SLOW／FAST／NORMAL

SET_OMEGA_SLOW／FAST／NORMAL　：　最大角速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_MAX_SLOW　(関節識別子) SET_OMEGA_MAX_FAST　(関節識別子) SET_OMEGA_MAX_NORMAL　(関節識別子)					
種別	汎用					
引数	脚識別子　：J11, J12, J13, . . . J43, ALL					
機能	指定された関節(ALL の場合は全ての関節)の角速度制限値(OMEGA_MAX)を一時的に変更する. SET_OMEGA_MAX_SLOW で, OMEGA_MAX の TIP_OMEGA_MAX_SLOW_RATE 倍(50%), SET_OMEGA_MAX_FAST で, OMEGA_MAX の TIP_OMEGA_MAX_FAST_RATE 倍(150%), SET_OMEGA_MAX_NORMAL で, OMEGA_MAX に戻す(100%).					
注意						
例	SET_OMEGA_MAX_FAST　J11 ：関節 J11 の角速度制限値を OMEGA_MAX の TIP_OMEGA_MAX_FAST 倍に設定する. SET_OMEGA_MAX _NORMAL ALL ：全ての関節の角速度制限値を各 OMEGA_MAX に戻す.					
See	SET_OMEGA_MAX, SET_OMEGA_MAX_RATE					
メモリ格納形式	Cmd	SET_OMEGA_MAX_SLOW, SET_OMEGA_MAX_FAST, SET_OMEGA_MAX_NORMAL				
	Target	関節識別子(J11, J12, J13,..., J43, ,ALL)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-2-6　SET_OMEGA_DOT_SLOW／FAST／NORMAL

SET_OMEGA_DOT_SLOW／FAST／NORMAL　：　関節角速度の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_OMEGA_DOT_MAX_SLOW　(関節識別子) SET_OMEGA_DOT_MAX_FAST　(関節識別子) SET_OMEGA_DOT_MAX_NORMAL　(関節識別子)					
種別	汎用					
引数	脚識別子　：J11, J12, J13, . . . J43, ALL					
機能	指定された関節(ALL の場合は全ての関節)の角加速度制限値(OMEGA_DOT_MAX)を一時的に変更する. SET_OMEGA_DOT_MAX_SLOW で OMEGA_DOT_MAX の TIP_OMEGA_DOT_MAX_SLOW_RATE 倍(50%), SET_OMEGA_DOT_MAX_FAST で OMEGA_DOT_MAX の TIP_OMEGA_DOT_MAX_FAST_RATE 倍(150%), SET_OMEGA_DOT_MAX_NORMAL で, OMEGA_DOT_MAX に戻す(100%).						
注意						
例	SET_OMEGA_DOT_MAX_FAST　J11 ：関節 J11 の角加速度制限値を OMEGA_DOT_MAX の TIP_OMEGA_DOT_MAX_FAST 倍に設定する. SET_OMEGA_DOT_MAX_NORMAL ALL ：全ての関節の角加速度制限値を各 OMEGA_DOT_MAX に戻す.					
See	**SET_OMEGA_DOT_MAX, SET_OMEGA_DOT_MAX_RATE**					
メモリ格納形式	Cmd	SET_OMEGA_DOT_MAX_SLOW, SET_OMEGA_DOT_MAX_FAST, SET_OMEGA_MAX_DOT_NORMAL				
	Target	関節識別子(J11, J12, J13,..., J43, ,ALL)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-2-7 SET_JUST_ANGLE

SET_JUST_ANGLE　:　関節角度制御時の許容誤差の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_JUST_POSITION （脚識別子/関節識別子/ALL） 指令値					
種別	汎用					
引数	脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ 関節識別子　　：Ｊ１１，Ｊ１２，Ｊ１３，．．． 指令値　　：関節型変数 ：定数あるいは汎用変数 正の数字．単位は[deg]．					
機能	指定された脚／関節の角度制御許容誤差(INPOS)を一時的に変更する．					
注意	指令値が負，あるいは適切な型ではない場合はエラー(Domain Error)とする．					
例	SET_JUST_ANGLE　F1　5.0 5.0 5.0 ：脚 F1 の角度制御許容誤差を 5.0[deg]に設定する． SET_JUST_ANGLE　ALL JNT1 ：全ての関節の角度制御許容誤差を JNT1(=JNT1.J1,...)の値に設定する． SET_JUST_ANGLE ALL 3.0 ：全ての間接の角度制御許容誤差を 3.0 に設定する．					
See	SET_NEAR_ANGLE, PTP_ANGLE, REL_PTP_ANGLE, COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE, WAIT_INPOS					
メモリ格納形式	Cmd	SET_JUST_ANGLE				
	Target	脚識別子(F1,F2,F3,F4,J11, J12,..., ALL)				
	ArgType1	指令値の種類				
	ArgType2	指令値の種類（脚識別子または ALL の場合は ARG_NONE）				
	ArgType3	指令値の種類（脚識別子または ALL の場合は ARG_NONE）				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	指令値の値（変数の場合は添え字，脚識別子または ALL の場合は 0.0）				
	Value3	指令値の値（変数の場合は添え字，脚識別子または ALL の場合は 0.0）				

7-8-2-8 SET_NEAR_ANGLE

<table>
<tr><th colspan="7">SET_NEAR_ANGLE　：　関節角度制御時の近傍到着許容誤差の変更</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>○</td></tr>
<tr><td>書式</td><td colspan="6">SET_NEAR_ANGLE （脚識別子） 指令値</td></tr>
<tr><td>種別</td><td colspan="6">汎用</td></tr>
<tr><td>引数</td><td colspan="6">脚識別子　　：Ｆ１，Ｆ２，Ｆ３，Ｆ４，ＡＬＬ
関節識別子　　：Ｊ１１，Ｊ１２，Ｊ１３，．．．
指令値　　：関節型変数
：定数あるいは汎用変数
正の数字．単位は[deg]．</td></tr>
<tr><td>機能</td><td colspan="6">指定された脚／関節の角度制御近傍誤差(NEAR)を一時的に変更する．</td></tr>
<tr><td>注意</td><td colspan="6">指令値が負，あるいは適切な型ではない場合はエラー(Domain Error)とする．</td></tr>
<tr><td>例</td><td colspan="6">SET_NEAR_ANGLE　F1　5.0 5.0 5.0
：脚 F1 の全関節の目標角度近傍誤差を 5.0[deg]に設定する．
SET_NEAR_ANGLE　ALL JNT1
：全ての関節の目標角度近傍誤差を JNT1(=JNT1.J1,...)の値に設定する．
SET_NEAR_ANGLE ALL 3.0
：全ての間接の角度制御近傍誤差を 3.0 に設定する．</td></tr>
<tr><td>See</td><td colspan="6">SET_JUST_ANGLE, PTP_ANGLE, REL_PTP_ANGLE, COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE, WAIT_NEAR</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">SET_NEAR_ANGLE</td></tr>
<tr><td>Target</td><td colspan="5">脚識別子(F1,F2,F3,F4,ALL), 関節識別子(J11, J12, J13,...)</td></tr>
<tr><td>ArgType1</td><td colspan="5">指令値の種類</td></tr>
<tr><td>ArgType2</td><td colspan="5">指令値の種類（脚識別子または ALL の場合は ARG_NONE）</td></tr>
<tr><td>ArgType3</td><td colspan="5">指令値の種類（脚識別子または ALL の場合は ARG_NONE）</td></tr>
<tr><td>Value1</td><td colspan="5">指令値の値（変数の場合は添え字）</td></tr>
<tr><td>Value2</td><td colspan="5">指令値の値（変数の場合は添え字，脚識別子または ALL の場合は 0.0）</td></tr>
<tr><td>Value3</td><td colspan="5">指令値の値（変数の場合は添え字，脚識別子または ALL の場合は 0.0）</td></tr>
</table>

7-8-3-1 SET_TORQUE_MAX

SET_TORQUE_MAX : 関節トルク制御時の最大関節トルクの変更						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_TORQUE_MAX (関節識別子) 指令値					
種別	汎用					
引数	関節識別子 : J11, J12, J13, ..., J43, ALL 指令値 : 定数あるいは汎用変数. 正の数字. 単位は[Nm].					
機能	指定された関節のトルク制限値(TORQUE_MAX)を一時的に変更する.					
注意	指令値が負, あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_TORQUE_MAX J11 50.0 : 関節 J11 のトルク制限値を 50.0[Nm]に設定する. SET_TORQUE_MAX J33 VAR03 : 関節 J33 のトルク制限値を VAR03 の値に設定する.					
See	SET_TORQUE_SOFT, SET_TOQRUE_FIRMLY, SET_TORQUE_NORMAL					
メモリ格納形式	Cmd	SET_TORQUE_MAX				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値 (変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-3-2 SET_TORQUE_MAX_RATE

SET_TORQUE_MAX_RATE : 最大関節トルクを比で変更						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_TORQUE_MAX_RATE (関節識別子) 指令値					
種別	汎用					
引数	関節識別子 : J11, J12, J13, ..., J43, ALL 指令値 : 定数あるいは汎用変数. 正の数字. 単位は[%].					
機能	指定された関節のトルク制限値(TORQUE_MAX)を一時的に(指令値)倍する.					
注意	指令値≦０, 指令値＞100, あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする.					
例	SET_TORQUE_MAX_RATE J11 50.0 : 関節 J11 のトルク制限値を 50.0[%]に設定する. SET_TORQUE_MAX_RATE J33 VAR03 : 関節 J33 のトルク制限値を VAR03 倍に設定する.					
See	SET_TORQUE_MAX, SET_TORQUE_MAX_SOFT / FIRMLY / NORMAL					
メモリ格納形式	Cmd	SET_TORQUE_MAX_RATE				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値 (変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-8-3-3　SET_TORQUE_MAX_FIRMLY／SOFT／NORMAL

SET_TORQUE_MAX_FIRMLY／SOFT／NORMAL　：　最大関節トルクの変更						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_TORQUE_MAX_FIRMLY　(関節識別子) SET_TORQUE_MAX_SOFT　(関節識別子) SET_TORQUE_MAX_NORMAL　(関節識別子)					
種別	汎用					
引数	脚識別子　：J11, J12, J13, . . . J43, ALL					
機能	指定された関節(ALL の場合は全ての関節)のトルク制限値(TORQUE_MAX)を一時的に変更する. SET_TORQUE_MAX_SOFT で, TORQUE_MAX の TIP_TORQUE_MAX_SLOW_RATE 倍(50%), SET_TORQUE_MAX_FIRMLY で, TORQUE_MAX の TIP_TORQUE_MAX_FAST_RATE 倍(150%), SET_TORQUE_MAX_NORMAL で, TORQUE_MAX に戻す(100%).					
注意						
例	SET_TORQUE_MAX_FAST　J11 ：関節 J11 のトルク制限値を TORQUE_MAX の TIP_TORQUE_MAX_FAST 倍に設定する. SET_TORQUE_MAX _NORMAL ALL ：全ての関節のトルク制限値を各 TORQUE_MAX に戻す.					
See	**SET_TORQUE_MAX, SET_TORQUE_MAX_RATE**					
メモリ格納形式	Cmd	SET_TORQUE_MAX_FIRMLY, SET_TORQUE_MAX_SOFT, SET_TORQUE_MAX_NORMAL				
	Target	関節識別子(J11, J12, J13,..., J43, ,ALL)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-3-4　SET_ANGLE_STIFFNESS

SET_ANGLE_STIFFNESS　:　関節剛性の設定						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_ANGLE_STIFFNESS （関節識別子） 指令値					
種別	汎用					
引数	関節識別子　　：J11, J12, J13, ..., J43, ALL 指令値　　：定数あるいは汎用変数．正の数字．単位は[mNm/deg]．					
機能	指定された関節の COMPLIANCE_ANGLE/REL_COMPLIANCE_ANGLE コマンド実行時の関節剛性を一時的に変更する．					
注意	指令値が負，あるいは定数か汎用変数ではない場合はエラー(Domain Error)とする．					
例	SET_ANGLE_STIFFNESS　J11　50.0 ：関節 J11 の関節剛性を 50.0[Nm/deg]に設定する． SET_ANGLE_STIFFNESS　J33 VAR03 ：関節 J33 の関節剛性を VAR03 の値に設定する．					
See	COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE					
メモリ格納形式	Cmd	SET_ANGLE_STIFFNESS				
	Target	関節識別子(J11, J12, J13, ..., J43, ALL)				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-3-5 SET_POSITION_STIFFNESS

SET_POSITION_STIFFNESS　:　バネの剛性の設定						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_POSITION_STIFFNESS （指識別子） 指令値 x 指令値 y 指令値 z					
種別	汎用					
引数	指識別子　　：F1, F2, F3, ALL 指令値 x,y,z　：定数あるいは汎用変数. 正の数字. 単位は[N/mm].					
機能	指定された指の COMPLIANCE_POSITION/REL_COMPLIANCE_POSITION コマンド実行時の関節剛性を一時的に変更する. ターゲットが指先でもｎ関節目でも共用する. 関節位置(J13, J23, J33)における COMPLIANCE_POSITION/REL_COMPLIANCE_POSITION にも同じバネ定数(k_spring)が使用される.					
注意	指令値が負, あるいは許される変数以外の型の場合はエラー(Domain Error)とする. z 成分は無視される（HiroshimaHnad).					
例	SET_POSITION_STIFFNESS　F1　50.0 60.0 0.0 ：指 1 のバネの剛性を(50.0, 60.0) [N/mm]に設定する. SET_POSITION_STIFFNESS　F1 VAR10 VAR20 0.0 ：指 1 のバネの剛性を(VAR10, VAR20)に設定する. ただし, z 成分(=0.0)は無視される(HiroshimaHand)					
See	COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION					
メモリ格納形式	Cmd	SET_POSITION_STIFFNESS				
	Target	指識別子(F1,F2, F3, ALL)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類				
	ArgType3	enum argument_typeT;　第三引数の種類				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字)				
	Value3	第三引数の値（変数の時は添え字)				

7-8-3-6 SET_COMPLIANCE_TORQUE_LIMIT

SET_COMPLIANCE_TORQUE_LIMIT　:　コンプライアンス制御時のトルク上限						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_COMPLIANCE_TORQUE_LIMIT （指／関節識別子） 指令値					
種別	汎用					
引数	指識別子　　：F1, F2, F3, ALL 関節識別子　　：J11, J12, J13,... 指令値　　：指識別子の場合には，定数，汎用型変数，または関節型変数. 関節識別子の場合には，定数あるいは汎用型変数のみ. 正の数字．単位は[mNm].					
機能	関節コンプライアンス制御時に有効になる．関節コンプライアンス制御は，目標角度と現在の角度との差に関節剛性(SET_ANGLE_STIFFNESS)を掛けたトルクを目標トルクとするが，この目標トルクに上限を設ける．上限を解除するには，RESET_COMPLIANCE_TORQUE_LIMIT を用いる.					
注意	**指令値がゼロ，あるいは負の場合はエラー(Domain Error)とする.** **トルク制御時など，コンプライアンス制御時以外の制御モードでのトルクを制限するものではない.**					
例	SET_COMPLIANCE_TORQUE_LIMIT　F1　50.0 60.0 10.0 ：指 1 の（コンプライアンス制御時の）目標トルクを J11=50, J12=60.0, J13=10 に制限. SET_COMPLIANCE_TORQUE_LIMIT　F1 JNT1 ：指 1 の目標トルクを J11=JNT1.J1, J12=JNT1.J2, J13=JNT.J3 に制限する. SET_COMPLIANCE_TORQUE_LIMIT　ALL 10.0 ：全関節の目標トルクを 10.0 に制限する.					
See	**COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE, SET_ANGLE_STIFFNESS, RESET_COMPLIANCE_TORQUE_LIMIT**					
メモリ格納形式	**Cmd**	SET_COMLIANCE_TOQRUE_LIMIT				
	Target	指識別子(F1,F2, F3, ALL), 関節識別子(J11, J12, J13,...)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類				
	ArgType3	enum argument_typeT;　第三引数の種類				
	Value1	第一引数の値（変数の時は添え字）				
	Value2	第二引数の値（変数の時は添え字）				
	Value3	第三引数の値（変数の時は添え字）				

7-8-3-7 RESET_COMPLIANCE_TORQUE_LIMIT

RESET_COMPLIANCE_TORQUE_LIMIT：コンプライアンス制御時のトルク上限を解除						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	RESET_COMPLIANCE_TORQUE_LIMIT （指／関節識別子）					
種別	汎用					
引数	指識別子　　：F1, F2, F3, ALL 関節識別子　　：J11, J12, J13,...					
機能	SET_COMPLIANCE_TORQUE_LIMIT で設定された関節コンプライアンス制御時のトルク上限を解除する.					
注意						
例	RESET_COMPLIANCE_TORQUE_LIMIT F1 ：指１のトルク制限を解除する. RESET_COMPLIANCE_TORQUE_LIMIT J11 ：指１関節１のトルク制限を解除する. RESET_COMPLIANCE_TORQUE_LIMIT ALL ：全関節のトルク制限を解除する.					
See	**COMPLIANCE_ANGLE, REL_COMPLIANCE_ANGLE, SET_ANGLE_STIFFNESS, SET_COMPLIANCE_TORQUE_LIMIT**					
メモリ格納形式	Cmd	RESET_COMLIANCE_TOQRUE_LIMIT				
	Target	指識別子(F1,F2, F3, ALL), 関節識別子(J11, J12, J13,...)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-3-8 SET_COMPLIANCE_FORCE_LIMIT

SET_COMPLIANCE_FORCE_LIMIT　：　コンプライアンス制御時の接触力の上限						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	SET_COMPLIANCE_FORCE_LIMIT （指識別子） 指令値					
種別	汎用					
引数	指識別子　　：F1, F2, F3, ALL 関節識別子　　：J13, J23, J33 (COMPLIANCE_POSITION コマンドが適用できる関節のみ) 指令値　　：定数，汎用型変数，または座標型変数. 正の数字．単位は[N].					
機能	コンプライアンス制御時に有効になる．コンプライアンス制御は，目標位置と現在の位置との差に剛性(SET_POSITION_STIFFNESS)を掛けた力を目標力とするが，この目標力に上限を設ける．上限を解除するには，RESET_COMPLIANCE_FORCE_LIMIT を用いる.					
注意	**指令値がゼロ，あるいは負の場合はエラー(Domain Error)とする.** **力制御時など，コンプライアンス制御時以外の制御モードでの力を制限するものではない.** 関節識別子は，指先位置ではなく，その関節位置（角度ではない）をコンプライアンス制御できる関節に限る.					
例	SET_COMPLIANCE_FORCE_LIMIT　F1　50.0 60.0 10.0 ：指 1 の（コンプライアンス制御時の）目標力を(X,Y,Z)=(50.0,60.0,10.0)に制限. SET_COMPLIANCE_FORCE_LIMIT　F1 POS1 ：指 1 の目標力を(X,Y,Z)=(POS1.X, POS1.Y, POS1.Z)に制限する. SET_COMPLIANCE_FORCE_LIMIT　ALL 10.0 10.0 10.0 ：全指／関節の目標力を(X,Y,Z)=(10.0, 10.0, 10.0)に制限する.					
See	**COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION, SET_POSITION_STIFFNESS, RESET_COMPLIANCE_FORCE_LIMIT**					
メモリ格納形式	**Cmd**	SET_COMLIANCE_FORCE_LIMIT				
	Target	指識別子(F1,F2, F3, ALL), 関節識別子(J13, J23, J33)				
	ArgType1	enum argument_typeT;　第一引数の種類				
	ArgType2	enum argument_typeT;　第二引数の種類				
	ArgType3	enum argument_typeT;　第三引数の種類				
	Value1	第一引数の値（変数の時は添え字)				
	Value2	第二引数の値（変数の時は添え字)				
	Value3	第三引数の値（変数の時は添え字)				

7-8-3-9 RESET_COMPLIANCE_FORCE_LIMIT

RESET_COMPLIANCE_FORCE_LIMIT：コンプライアンス制御時の接触力上限を解除						
対応	TitanVIII	×	HiroshimaHand	○	汎用	○
書式	RESET_COMPLIANCE_FORCE_LIMIT （指／関節識別子）					
種別	汎用					
引数	指識別子　　：F1, F2, F3, ALL 関節識別子　　：J13, J23, J33					
機能	SET_COMPLIANCE_FORCE_LIMIT で設定されたコンプライアンス制御時の目標接触力の上限を解除する.					
注意	関節識別子は，指先位置ではなく，その関節位置（角度ではない）をコンプライアンス制御できる関節に限る.					
例	RESET_COMPLIANCE_FORCE_LIMIT F1 ：指１の接触力制限を解除する. RESET_COMPLIANCE_FORCE_LIMIT ALL ：全指／関節のトルク制限を解除する.					
See	**COMPLIANCE_POSITION, REL_COMPLIANCE_POSITION, SET_POSITION_STIFFNESS, SET_COMPLIANCE_FORCE_LIMIT**					
メモリ格納形式	**Cmd**	RESET_COMLIANCE_FORCE_LIMIT				
	Target	指識別子(F1,F2, F3, ALL), 関節識別子(J11, J12, J13,...)				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-8-4-1 SET_SAMPLING_TIME

SET_SAMPLING_TIME　：　サンプリングタイムの変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_ANGLE_STIFFNESS　サンプリングタイム					
種別	汎用					
引数	サンプリングタイム：　制御サイクル，単位はミリ秒[ms]					
機能	ロボットの制御サイクルを変更する．汎用変数も可 単位は異なるが，機能的には SET_SAMPLING_FREQ と同一					
注意	サンプリングタイムが範囲外（含む負）の場合はエラー(Domain Error)とする.					
例	SET_SAMPLING_TIME　50.0 ：サンプリングタイムを 50[ms]に設定する. SET_SAMPLING_TIME VAR03 ：サンプリングタイムを VAR03 の値に設定する.					
See	SET_SAMPLING_FREQ					
メモリ格納形式	Cmd	SET_SAMPING_TIME				
	Target	NO_TARGET				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-8-4-2 SET_SAMPLING_FREQ

SET_SAMPLING_FREQ　：　サンプリング周期の変更						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	SET_ANGLE_STIFFNESS　サンプリング周期					
種別	汎用					
引数	サンプリング周期：　制御サイクル，単位は[Hz]					
機能	ロボットの制御サイクルを変更する．汎用変数も可. 単位は異なるが，機能的には SET_SAMPLING_TIME と同一					
注意	サンプリング周期が範囲外（含む負）の場合はエラー(Domain Error)とする.					
例	SET_SAMPLING_TIME　50.0 ：サンプリング周期を 50[Hz]に設定する. SET_SAMPLING_TIME VAR03 ：サンプリング周期を VAR03 の値に設定する.					
See	SET_SAMPLING_TIME					
メモリ格納形式	Cmd	SET_SAMPING_FREQ				
	Target	NO_TARGET				
	ArgType1	指令値の種類				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	指令値の値（変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-9 入出力系コマンド

7-9-1　AD_IN

AD_IN　：　アナログ入力						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	AD_IN （ボード番号）（チャンネル番号）（汎用変数）					
種別	準汎用					
引数	ボード番号　　：　０～ チャンネル番号：　０～ｎ　（ｎはボード番号によって異なる） 汎用変数　　：　VARnn ボード番号，チャンネル番号の代わりに「Ｉ／Ｏ識別子」による選択も可能. Ｉ／Ｏ識別子　：ＶＲ１，ＶＲ２，ＶＲ３ Ｊ１１，Ｊ１２，Ｊ１３，Ｊ２１，．．．，Ｊ４３ ＰＡＷ１，ＰＡＷ２，ＰＡＷ３，ＰＡＷ４ ＡＮＴＥＮＮＡ					
機能	AD 変換ボードから電圧[V]を読み込み，汎用変数にセットする					
注意	現在は，ＡＤ変換ボードが１枚のみなので，ボード番号は０のみ，チャンネル番号は０～１５のみ．ボード番号，チャンネル番号共に変数使用不可．理由は面倒な割には利用されることが少ないから					
例	AD_IN　0 13 VAR03 ：13ch(VR1)の電圧を汎用変数 VAR03 に読み込む. AD_IN VR1 VAR03 ：VR1 の電圧を汎用変数 VAR03 に読み込む.					
See						
メモリ格納形式	Cmd	AD_IN				
	Target	関節識別子(J11～J43), I/O 識別子(VR1～VR3), NONE				
	ArgType1	CONSTANT（ボード番号), NONE (I/O 識別子指定時)				
	ArgType2	CONSTANT（チャンネル番号), NONE (I/O 識別子指定時)				
	ArgType3	VAR				
	Value1	ボード番号，ただし I/O 識別子指定時は 0.0				
	Value2	チャンネル番号，ただし I/O 識別子指定時は 0.0				
	Value3	汎用変数の添え字				

7-9-2　KEY_IN

KEY_IN　：　キーボードからの１文字入力						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	KEY_IN （汎用変数）					
種別	汎用					
引数	汎用変数　　：　VARnn　（省略可能）					
機能	キーボードから１文字(1byte)入力されるのを待つ．入力された文字の文字コードが汎用変数にセットされる．この時にアルファベット小文字は大文字に変換される．引数が省略された場合は，単にキー入力を待つ．RS232C ポートしか持たない場合は RS232C からの入力を待つ． KEY_IN コマンド実行後，キーボードバッファは完全に空になるまで空読みされる．これは安全のための措置である．したがって，連続的にキーボードから文字列を取り込みたいと考えた場合は，多少の工夫が必要である．					
注意	このコマンドが有効な間だけ，[RETURN]キー（＝ソフトウェア非常停止）および[ESC]キー（＝サイクル停止要求）以外のキー入力はシステム側に通知されない．					
例	KEY_IN VAR03 ：キーボードからの入力(1byte)を VAR03 に読み込む． KEY_IN ：キーボードからの入力を待つ．					
See	MESSAGE, FLUSH_BUF, KB_HIT(イベントフラグ)					
メモリ格納形式	Cmd	KEY_IN				
	Target	NO_TARGET				
	ArgType1	VAR，または ARG_NONE（引数未指定時）				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	汎用変数の添え字，または 0.0（引数未指定時）				
	Value2	0.0				
	Value3	0.0				

7-9-3　FLUSH_BUF

FLUSH_BUF　：　キーボードバッファのクリア						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	FLUSH_BUF					
種別	汎用					
引数	なし					
機能	キーボードバッファ（RS232C 受信バッファ）を空にする．主に，イベントフラグ KB_HIT を使用する前に，キーボードバッファを空にするのに用いる．					
注意						
例	FLUSH_BUF L1: IF KB_HIT GOTO L2: GOTO L1: L2:　/* ただの入力待ちのサスペンドループならば，KEY_IN コマンド１行で構わない　*/					
See	**KB_HIT(イベントフラグ), KEY_IN**					
メモリ格納形式	Cmd	FLUSH_BUF				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

（参考）アスキーコード表

32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	″	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	110	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	I	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92		108	l	124	
45	−	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

緑色のマスの文字は KEY_IN コマンドにより変数に読み込み可能と思われる．その場合のコードが，左側のグレーのマスの中の数字（１０進数）である．小文字は大文字に変換される．

７-９-４　MESSAGE

MESSAGE　：　文字列の画面表示						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	**MESSAGE （文字列）**					
種別	汎用					
引数	文字列　　：　全角も可．スペースを含んでも良い． ただし長さは TIP_MAX_MESSAGE_LENGTH(70byte)文字以内．					
機能	ＰＣ画面上の文字表示領域（１行のみ）にメッセージを表示する． ¥a でベル，¥t でタブ．RS232C ポートしか持たない場合は，RS232C ポートへ出力する．文字列が長すぎたときは，エラー(too much long string)とする．文字列中に以下の記述があった場合，対応する汎用変数の値と置き換えて表示する．ただし，指定された汎用変数に値が代入されていない場合は，”Possible use of Variable before definition”を表示して非常停止する． （汎用変数値への置換の書式）[11] %汎用変数名%　：汎用変数の値（小数点以下切り捨て）に置換する． %汎用変数名.n% ： n は小数点以下の桁数（0 ＜ n ＜ 8）． %と%の間に余分な文字が混ざっていた場合または該当する汎用変数名が存在しない場合は無効とし，汎用変数への置き換えを行わない．					
注意	メッセージ中に；（半角セミコロン）を入れるとマルチステートメントと解釈され，エラーとなってしまいます． どうしてもセミコロンを使いたい場合は全角のセミコロンを使ってください．					
例	MESSAGE　何かキーを押してください．　(Q:終了) MESSAGE ：行クリア（MESSAGE_CLEAR と等価） (1) MESSAGE　斜面の角度は%HEIGHT.0%[deg]です． (2) MESSAGE　斜面の角度は%HEIGHT.3%[deg]です． HEIGHT = 32.5 の場合， (1)は，斜面の角度は 32[deg]です． (2)は，斜面の角度は 32.500[deg]です． MESSAGE これは失敗例%HEIGHT.3 %です．（余分なスペースが入っている）					
See	**KEY_IN, MESSAGE_CLEAR**					
メモリ格納形式	Cmd	MESSAGE				
	Target	NO_TARGET				
	ArgType1	STRING				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	文字列テーブルの番号				
	Value2	0.0				
	Value3	0.0				

[11] Ver.3.0 以前とは大きく変わっているので注意すること．以前は，%fnn%または%dnn%という書式だったが，自由な変数名を導入した結果，番号(nn)で指定することができなくなったため，仕様を大きく変えた．また，近いうちに JNT 型，POS 型変数にも対応予定．その場合には精度を表す'.'が別の文字に変わるものと思われる．

7-9-5 MESSAGE_CLEAR

MESSAGE_CLEAR　：　文字列表示領域のクリア						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	MESSAGE_CLEAR					
種別	汎用					
引数	なし					
機能	ＰＣ画面上の文字表示領域（１行のみ）をクリアする.					
注意						
例	MESSAGE_CLEAR					
See	MESSAGE					
メモリ格納形式	Cmd	MESSAGE_CLEAR				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-9-6 LAMP_ON

LAMP_ON　：　外部 ON/OFF 出力への ON 指令						
対応	TitanVIII	○	HiroshimaHand	○	汎用	△
書式	LAMP_ON （デバイス番号）					
種別	汎用					
引数	デバイス番号：　０～２５５の整数（小数点以下切り捨て）					
機能	引数で指定された表示灯（あるいはリレー，ソレノイドなど）を ON にする.					
注意	全てのデバイスを ON する ALL という引数も考えたが，危険なので仕様には入れない. デバイス番号が不正な場合は実行時に Domain error を発生し，非常停止する．値の範囲や値が適正かの判断は，機種固有の情報のため tip.cpp ではなく，tip_if.cpp で行うため.					
例	LAMP_ON 1					
See	**LAMP_OFF**					
メモリ格納形式	Cmd	LAMP_ON				
	Target	NO_TARGET				
	ArgType1	CONST または VAR				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	即値，または汎用変数の添え字				
	Value2	0.0				
	Value3	0.0				

7-9-7 LAMP_OFF

LAMP_OFF　：　外部 ON/OFF 出力への OFF 指令						
対応	TitanVIII	○	HiroshimaHand	○	汎用	△
書式	LAMP_ON （デバイス番号）					
種別	汎用					
引数	デバイス番号：　０～２５５の整数（小数点以下切り捨て）					
機能	引数で指定された表示灯（あるいはリレー，ソレノイドなど）を OFF にする.					
注意	全てのデバイスを ON する ALL という引数も考えたが，危険なので仕様には入れない. デバイス番号が不正な場合は実行時に Domain error を発生し，非常停止する．値の範囲や値が適正かの判断は，機種固有の情報のため tip.cpp ではなく，tip_if.cpp で行うため.					
例	LAMP_OFF 1					
See	**LAMP_ON**					
メモリ格納形式	Cmd	LAMP_OFF				
	Target	NO_TARGET				
	ArgType1	CONSTANT または VAR				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	即値，または汎用変数の添え字				
	Value2	0.0				
	Value3	0.0				

7-9-8 SILENT_ON

SILENT_ON　：　画面表示を最小限にする						
対応	TitanVIII	×	HiroshimaHand	○	汎用	△
書式	SILENT_ON					
種別	汎用					
引数	なし					
機能	画面上への情報表示を最小限にする．ディスプレイ画面をビデオ映像とスーパーインポーズ撮影する際に，モード表示やメッセージ表示などを消去したい場合に用いる．					
注意						
例	SILENT_ON					
See	SILENT_OFF, MESSAGE					
メモリ格納形式	Cmd	SILENT_ON				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-9-9 SILENT_OFF

SILENT_OFF　：　画面表示を元に戻す						
対応	TitanVIII	×	HiroshimaHand	○	汎用	△
書式	SILENT_OFF					
種別	汎用					
引数	なし					
機能	画面上への情報表示を元に戻す.					
注意						
例	SILENT_OFF					
See	SILENT_ON, MESSAGE					
メモリ格納形式	Cmd	SILENT_OFF				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

７-１０ その他

７-１０-１ DEF_VAR / DEF_POS / DEF_JOINT

DEF_VAR / DEF_POS / DEF_JOINT　：　変数の宣言						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	DEF_VAR　（変数名），（変数名），．．． DEF_POS　（変数名），（変数名），．．． DEF_JOINT　（変数名），（変数名），．．．					
種別	汎用					
引数						
機能	TIP プログラム内で使用する変数名を宣言する. 宣言されていない変数を使用しようとした場合には，エラー(unknown variable)を出力する.					
注意	中間言語ファイル(FORMAL.TMP)作成時に読み飛ばされるので，中間言語コードはメモリに格納されない.					
例	DEF_VAR　VAR10, SW, BODY_HEIGHT DEF_POS　POS10, TOP_POSITION_F1 DEF_JOINT INITIAL_ANGLE_F1					
See						
メモリ格納形式	メモリ中には格納されない.					

７-１０-２　REM

REM　：　注釈行						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	REM　（コメント）					
種別	汎用					
引数	コメント．全角／２バイト半角もＯＫ					
機能	TIP プログラムにコメントを付ける. 行注釈を入れるには，REM 以外にも"//"が使用できる. Ver.3.0 からコマンド扱いになった．１コマンドとして解釈されるため，１サイクル消費される．1 サイクル消費されるのを嫌う場合は//を用いるべきである．ＮＯＰの代用になるので，スキャンサイクルを厳密に計算したい場合には便利である.					
注意	.					
例	REM　このプログラムは＊＊年＊＊月に作成されました.					
See						
メモリ格納形式	Cmd	REM				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-3　END

END　：　TIP プログラムの終了						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	END					
種別	汎用					
引数	なし					
機能	（プログラム読み込み時） この行以降にもスクリプトが記述されていても無視される．また，１プログラム中には必ず一つの END が無いとエラーになる．１ファイル中にいくつ記述しても構わないので，動作デバッグ時にそれ以降の行を無視したい場合などにも重宝する． （実行時） TIP プログラムの実行を中断し，制御モードを手動モードに移す．					
注意						
例	END					
See	**EXIT**					
メモリ格納形式	Cmd	END				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-4　EXIT

EXIT　：　プログラムの強制終了						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	EXIT					
種別	汎用					
引数	なし					
機能	TIP プログラムの実行を中断し,手動モードに移行する.					
注意						
例	EXIT					
See	**END**					
メモリ格納形式	Cmd	EXIT				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-5　LOG_START

LOG_START　：　ログ記録開始						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	LOG_START					
種別	汎用					
引数	なし					
機能	サンプリングを開始する．もしも LOG_STOP で中断されていた場合は追加される． サンプリング結果は TIP プログラム終了後(END コマンド実行時)にファイル(TITAN.LOG)に書き出される．サンプリングはバッファ領域が一杯になるか，LOG_STOP が実行されるまで続けられる．					
注意	サンプリングレートは，SET_SAMPLING_TIME/FREQ コマンドで変更可能．					
例	LOG_START					
See	**LOG_STOP, LOG_ONE_SHOT, SET_SAMPLING_TIME/FREQ**					
メモリ格納形式	Cmd	LOG_START				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-6　LOG_STOP

LOG_STOP　：　ログ記録中断						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	LOG_STOP					
種別	汎用					
引数	なし					
機能	サンプリングを一時中断する．サンプリング結果は TIP プログラム終了後(END コマンド実行時)にファイル(TITAN.LOG)に書き出される．サンプリングはバッファ領域が一杯になるか，LOG_STOP が実行されるまで続けられる．					
注意	サンプリングレートは，SET_SAMPLING_TIME/FREQ コマンドで変更可能．					
例	LOG_STOP					
See	**LOG_START, LOG_ONE_SHOT, SET_SAMPLING_TIME/FREQ**					
メモリ格納形式	Cmd	LOG_STOP				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-7 LOG_ONE_SHOT

LOG_ONE_SHOT　：　現在の情報をログに記録						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	LOG_ONE_SHOT					
種別	汎用					
引数	なし					
機能	いま現在の各種センサ類の値を 1 サンプリング分だけ記録する．TEACH 系コマンドで変数に記録することはできるが，ファイルには書き出せないため．LOG_START だと，それ以降，垂れ流し式に記録されてしまう．LOG_START/LOG_STOP を連続して書けば同じ機能を実現できるが，記述が面倒であることや，将来的に複数コマンドを 1 サイクルで同時に実行する機能を搭載した場合に，START/STOP が同じサイクルで実行された結果，何も記録されないような事態が発生するのを避ける目的がある．確実に現在の値だけを記録したい（たとえばキーを押された瞬間）場合に用いる．					
注意						
例	LOG_ONE_SHOT					
See	**LOG_START, LOG_STOP**					
メモリ格納形式	Cmd	LOG_ONE_SHOT				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-8　OUTPUT_VARIABLES

OUTPUT_VARIABLES　：　全ての変数をファイルに書き出す						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	OUTPUT_VARIABLES					
種別	汎用					
引数	なし					
機能	変数の値を全てファイルに書き出す．安全のために，ロボットはその場停止(ALL_STOP)する．ファイルへの書きこみが終了したら，ALL_STOP_RELEASE される．ただし，もしこのコマンド以前に ALL_STOP が掛けられていた場合は，ALL_STOP_RELEASE されない．					
注意						
例	OUTPUT_VARIABLES					
See	ALL_STOP, ALL_STOP_RELEASE					
メモリ格納形式	Cmd	OUTPUT_VARIABLES				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-9　EMG_OFF

EMG_OFF　：　関節角度異常での非常停止を無視						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	EMG_OFF					
種別	汎用					
引数	なし					
機能	関節角度の非常停止を無視する．関節角度が LIMIT に到達した場合，エラーとはせずに，その場で停止する．ただしポテンショメータの LIMIT に到達した場合は非常停止を発生する．主に速度制御，角速度制御，トルク制御の際に非常停止に入ってしまうのを防ぐのが目的である．					
注意	**TIP モードの初期値は，EMG_ON**					
例	EMG_OFF					
See	**EMG_ON**					
メモリ格納形式	Cmd	EMG_OFF				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-10　EMG_ON

EMG_ON　：　関節角度異常時の非常停止を有効化						
対応	TitanVIII	○	HiroshimaHand	○	汎用	○
書式	EMG_ON					
種別	汎用					
引数	なし					
機能	EMG_OFF を解除する					
注意	**TIP モードの初期値は，EMG_ON**					
例	EMG_ON					
See	**EMG_OFF**					
メモリ格納形式	Cmd	EMG_ON				
	Target	NO_TARGET				
	ArgType1	ARG_NONE				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	0.0				
	Value2	0.0				
	Value3	0.0				

7-10-11　SPECIAL

SPECIAL　：　外部コマンドの呼び出し						
対応	TitanVIII	○	HiroshimaHand	○	汎用	△
書式	SPECIAL　(外部コマンド番号)　[引数1] [引数2]					
種別	汎用					
引数	外部コマンド番号（省略不可）：0以上255以下の整数（小数点以下は切り捨て） 引数1，2：外部コマンドに渡す値．即値あるいは汎用変数，省略可能					
機能	コマンド番号で指定された外部コマンドを実行する．実行ステップのカウントアップは外部コマンドが行う（tip_if.cpp）．					
注意						
例	SPECIAL 0 1.0					
See	SPECIAL_FLAG_n（イベントフラグ）					
メモリ格納形式	Cmd	SPECIAL				
	Target	NO_TARGET				
	ArgType1	CONSTANT				
	ArgType2	CONSTANT または VAR（省略時は ARG_NONE）				
	ArgType3	CONSTANT または VAR（省略時は ARG_NONE）				
	Value1	外部コマンド番号				
	Value2	即値または汎用変数の添え字（省略時は 0.0）				
	Value3	即値または汎用変数の添え字（省略時は 0.0）				

実装済みの SPECIAL コマンドについては，８章を参照してください．

7-１１ 特殊コマンド

7-１１-１　STAND_UP

STAND_UP　：　立ち上がる						
対応	TitanVIII	○	HiroshimaHand	×	汎用	×
書式	STAND_UP　(高さ)					
種別	Titan 特化					
引数	高さ[mm]，汎用変数可能．省略不可					
機能	立ち上がる．主に土台から立ち上がる時に利用する．全ての脚の制御モードを脚先位置制御モードに切り替え，現在の位置に位置決めする．位置決め完了後，各脚先位置の目標値のｚ座標を －（高さ)[mm]することで，Titan の重心を上方に(高さ)[mm]持ち上げる．変数はVARnn 変数のみ使用可能．速度パラメータは一時的に低速(**SET_V_MAX_SLOW**)に設定される．動作終了後に通常の速度(**SET_V_MAX_NORMAL**)に戻る．					
注意	トルク制御，コンプライアンス制御など，力要素を含む制御を行っていた場合は，エラーとする．					
例	STAND_UP　100.0 STAND_UP　VAR03 STAND_UP　-VAR03					
See	**SIT_DOWN**					
メモリ格納形式	**Cmd**	STAND_UP				
	Target	NO_TARGET				
	ArgType1	引数（高さ）の種類 (CONSTANT, VAR)				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	引数（高さ）の値（汎用変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

7-11-2　SIT_DOWN

SIT_DOWN　：　座る						
対応	TitanVIII	○	HiroshimaHand	×	汎用	×
書式	SIT_DOWN　(高さ)					
種別	Titan 特化					
引数	高さ[mm]，汎用変数可能．省略不可					
機能	腰を下ろす．主に土台から立ち上がり，土台を取り除いた後に元の姿勢に戻るのに利用する．精度の観点から考えると，INITIAL_POSTURE などを用いるべきか s も知れないが，STAND_UP した時の姿勢が INITIAL_POSTURE とは限らないため，用意したもの．全ての脚の制御モードを脚先位置制御モードに切り替え，現在の位置に位置決めする．位置決め完了後，各脚先位置の目標値の z 座標を +(高さ)[mm]することで，Titan の重心を下方に(高さ)[mm]下げる．速度パラメータは一時的に低速(**SET_V_MAX_SLOW**)に設定される．動作終了後に通常の速度(**SET_V_MAX_NORMAL**)に戻る．					
注意	トルク制御，コンプライアンス制御など，力要素を含む制御を行っていた場合は，エラーとする．					
例	SIT_DOWN　100.0 SIT_DOWN　VAR03 SIT_DOWN　-VAR03					
See	**STAND_UP**					
メモリ格納形式	Cmd	SIT_DOWN				
	Target	NO_TARGET				
	ArgType1	引数（高さ）の種類 (CONSTANT, VAR)				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	引数（高さ）の値（汎用変数の場合は添え字）				
	Value2	0.0				
	Value3	0.0				

7-11-3　DETECT_FLOOR

<table>
<tr><th colspan="7">DETECT_FLOOR　：　床を検出する</th></tr>
<tr><td>対応</td><td>TitanVIII</td><td>○</td><td>HiroshimaHand</td><td>○</td><td>汎用</td><td>×</td></tr>
<tr><td>書式</td><td colspan="6">DETECT_FLOOR　(脚識別子)（Ｔｙｐｅ）</td></tr>
<tr><td>種別</td><td colspan="6">TitanVIII / HiroshimaHand 特化</td></tr>
<tr><td>引数</td><td colspan="6">脚識別子： F1, F2, F3, F4, ALL

HiroshimaHand と TitanVIII とで動作が異なる.
(HiroshimaHand)
Type：１ = 第一関節のみトルク制御
Type：２ = 第一および第二関節をトルク制御
(TitanVIII)
Type：１ = (ALL の時は)脚１本ずつ順番に降ろして行く.
Type：２ = (ALL の時は)全ての脚を同時に降ろす.
Type：３ = (ALL の時は)全ての脚を同時に持ち上げる.
省略不可</td></tr>
<tr><td>機能</td><td colspan="6">(HiroshimaHand)
床を探す動作を行う．指定された関節以外の関節の絶対角度は変えずに，指定された指関節を一定トルク制御で閉じていく．全ての指が床を発見したら，その場で停止する.
(TitanVIII)
足裏の圧力センサを利用して，床を探索する．床を発見したら，その位置で脚先位置制御モードで停止する．Type1,2 は，空中に脚先が浮いた状態から床を探す．Type3 は，脚先が地面に着いた状態からボディーを降ろして（＝脚を持ち上げて），床から離れた脚を順次，停止させていく．脚先は，低速（V_MAX の比率を SLOW に設定）の速度制御で上下に動く．脚識別子で特定の脚が指定された場合は，Type1 も Type2 も同じ動作となる.</td></tr>
<tr><td>注意</td><td colspan="6">発見したか否かを判別するフラグは存在しないため，床が発見されなかった場合は次のステップに進めない.</td></tr>
<tr><td>例</td><td colspan="6">DETECT_FLOOR 1</td></tr>
<tr><td>See</td><td colspan="6">DETECT_OBJECT(HiroshimaHand のみ)</td></tr>
<tr><td rowspan="8">メモリ
格納形式</td><td>Cmd</td><td colspan="5">DETECT_FLOOR</td></tr>
<tr><td>Target</td><td colspan="5">脚識別子(F1, F2, F3, F4, ALL)</td></tr>
<tr><td>ArgType1</td><td colspan="5">引数の種類 (CONSTANT, VAR)</td></tr>
<tr><td>ArgType2</td><td colspan="5">ARG_NONE</td></tr>
<tr><td>ArgType3</td><td colspan="5">ARG_NONE</td></tr>
<tr><td>Value1</td><td colspan="5">引数の値 (汎用変数の場合は添え字)</td></tr>
<tr><td>Value2</td><td colspan="5">0.0</td></tr>
<tr><td>Value3</td><td colspan="5">0.0</td></tr>
</table>

7-１１-4 DETECT_OBJECT

DETECT_OBJECT　：　対象物を検出する						
対応	TitanVIII	×	HiroshimaHand	○	汎用	×
書式	DETECT_OBJECT　（Ｔｙｐｅ）					
種別	HiroshimaHand 特化					
引数	Type：１ = 全関節を制御　（指先の力制御）. Type：２ = 指先関節を除く（第２関節の力制御）. Type：３ = 全関節を制御　（指先のハイブリッド制御：バネ＋高さ維持） Type：４ = 指先関節を除く（第２関節のハイブリッド制御：バネ＋高さ維持）:未実装 省略不可					
機能	引数で指定されたモードに従って，対象物にアプローチする. 対象物を発見したら停止する．DETECT_FLOOR の後に使用するのが望ましい.					
注意	発見したか否かを判別するフラグは存在しないため，対象物が発見されなかった場合は次のステップに進めない.					
例	DETECT_OBJECT 1					
See	DETECT_FLOOR					
メモリ格納形式	Cmd	DETECT_OBJECT				
	Target	NO_TARGET				
	ArgType1	引数の種類（CONSTANT, VAR)				
	ArgType2	ARG_NONE				
	ArgType3	ARG_NONE				
	Value1	引数の値（汎用変数の場合は添え字)				
	Value2	0.0				
	Value3	0.0				

８ 実装済み SPECIAL コマンドリファレンス

SPECIAL コマンドの使用方法に関しては，７-１０-１１節を参照してください．
また，実装された SPECIAL コマンドの仕様は断り無く変更される可能性がありますので注意してください．外部コマンド番号に関しては，極力変更しないようにしますが，別のコマンドに置き換わる可能性がありますので，同様に注意してください．[12]

８-１ TitanVIII 用

実装された SPECIAL コマンドは存在しません．

８-２ HiroshimaHand 用

[一覧表]

番号	名称	説明
特殊な制御に関する SPECIAL コマンド		
1	Lever motion (初期化)	第２関節に横方向のバネを設定する． 初期姿勢の設定（COMPLIANCE_POSITION コマンドを追加したので，実質的に不要）
2	Lever motion	第２関節に横方向のバネを設定する． 実際の制御はこちら（COMPLIANCE_POSITION コマンドを追加したので，実質的に不要）．
5	第２種 AFC	Active Force Closure (原田氏) の実験用．対象物同士の相対位置を変化させることなく，複数対象物系の質量中心に任意の加速度，角加速度を発生させる．
Active Friction Control に関する SPECIAL コマンド		
10	AFC OFF	全ての指の振動を停止する．
11	指１の AFC ON	モードと振幅を設定し，振動開始
12	指２の AFC ON	
13	指３の AFC ON	
14	指１の振動周波数を設定	各指の振動数を設定する．
15	指２の振動周波数を設定	
16	指３の振動周波数を設定	
表示に関する SPECIAL コマンド		
20	BAR グラフ表示	バーグラフ１または２の表示を ON/OFF する．

[12] 使用頻度の高いコマンドは標準コマンドに格上げされるため．また仕様がハッキリ決められないために SPECIAL コマンドであるが故に，致し方ない．

8-2-1　特殊な制御に関する SPECIAL コマンド

番号	名称
1	Lever motion (初期化)
例）SPECIAL 1	
第 2 関節に横方向のバネを設定する． x 方向にはバネを設定して目標位置(ハンド中央)からの距離に比例した力を発生するようにコンプライアンス制御を行い，y 方向には SPECIAL 1 を実行した時の第 2 関節の高さを位置するように位置制御を行う．これら二つの制御目的を実現するように関節への制御出力を求めて速度制御を行う．ただし，指先関節のみ位置制御とし，この SPECIAL コマンドからは制御を行わない．	

	引数の種類	説明
引数 1	VAR	バネ定数 [N/m]
引数 2	なし	

番号	名称
2	Lever motion
例）SPECIAL 2 2.0	
実際に Lever motion を行うには，SPECIAL 2 を呼び続ける必要がある． Lever motion を終了するための SPECIAL コマンドは存在しない．PTP_REL_ANGLE コマンドなどを用いて強制的に制御モードを変えるだけでよい．	

	引数の種類	説明
引数 1	VAR	バネ定数 [N/m]
引数 2	なし	

番号	名称
5	第 2 種 Active Force Closure (初期化)
例）SPECIAL 5 1.0	
定常トルク（例えば初期トルク），フィードバックゲインを設定する．また，θd の初期値を記憶する．設定した変位量を動いたら，SPECIAL_FLAG0 が ON になる．	

	引数の種類	説明
引数 1	VAR	トルク制御フィードバックゲイン (mNm/deg)
引数 2	VAR	変位量[mm]

番号	名称
6	第 2 種 Active Force Closure
例）SPECIAL 6 1.0 -5.0	
第 2 種 ActiveForceClosure を実行するには，SPECIAL 5 をコールし続ける必要がある． x,y 方向の加速度を発生する．対象物の重心位置，接触点などはプログラム中に直に記述されるので，その都度，変更してコンパイルし直す必要がある．第 2 種 AFC を終了するための SPECIAL コマンドは存在しない．PTP_REL_ANGLE コマンドなどを用いて強制的に制御モードを変えるだけでよい．	

	引数の種類	説明
引数 1	VAR	x 方向の速度 ([mm/sec])
引数 2	VAR	y 方向の速度 ([mm/sec])

８-２-２　ActiveFrictionControl 関連の SPECIAL コマンド

モードの説明：

0：停止　(mode = 0, on = false)
1：指先関節のみ振動　([V])
2：全関節の振動　(同一方向[V])
3：全関節の振動　(緩める方向[V])
4：指１のみによる Rolling　(CW)
5：全指による Rolling　(CW)　：nofunction
6：指１のみによる Rolling　(CCW)
7：全指による Rolling　(CCW)　：nofunction

番号	名称	
10	AFC OFF	
例）SPECIAL 10		
全ての指の振動を停止する． 特定の指の振動を停止するには，SPECIAL 11,12,3 を用いる（モードに０を指定）．		
	引数の種類	説明
引数１	なし	
引数２	なし	

番号	名称	
11	指 1 の AFC ON	
例）SPECIAL 11.0 4.0 2.0		
指 1 を指定されたモード，振幅で振動させる． モードに 0 を指定すれば，振動を停止することができる． 確実に全ての指の振動を停止したい場合は，SPECIAL 10 を用いる．		
	引数の種類	説明
引数 1	VAR	モード （0～7）
引数 2	VAR	振幅．単位は各モードによって異なる． ただし，現在実装されているモードでは，[V]のみ．

番号	名称	
12	指 2 の AFC ON	
例）SPECIAL 12.0 4.0 2.0		
指 2 を指定されたモード，振幅で振動させる． モードに 0 を指定すれば，振動を停止することができる． 確実に全ての指の振動を停止したい場合は，SPECIAL 10 を用いる．		
	引数の種類	説明
引数 1	VAR	モード （0～7）
引数 2	VAR	振幅．単位は各モードによって異なる． ただし，現在実装されているモードでは，[V]のみ．

番号	名称	
13	指 3 の AFC ON	
例）SPECIAL 13.0 4.0 2.0		
指 3 を指定されたモード，振幅で振動させる． モードに 0 を指定すれば，振動を停止することができる． 確実に全ての指の振動を停止したい場合は，SPECIAL 10 を用いる．		
	引数の種類	説明
引数 1	VAR	モード （0～7）
引数 2	VAR	振幅．単位は各モードによって異なる． ただし，現在実装されているモードでは，[V]のみ．

番号	名称	
14	指 1 の振動周波数を設定	
例）SPECIAL 14 30.0		
指 1 の振動数を設定するが，振動を ON/OFF する機能は無い．振動を ON/OFF には，SPECIAL 11 を実行する．		
	引数の種類	説明
引数 1	VAR	振動数［Hz］
引数 2	なし	

番号	名称	
15	指 2 の振動周波数を設定	
例）SPECIAL 15 30.0		
指 2 の振動数を設定するが，振動を ON/OFF する機能は無い．振動を ON/OFF には，SPECIAL 12 を実行する．		
	引数の種類	説明
引数 1	VAR	振動数［Hz］
引数 2	なし	

番号	名称	
16	指 3 の振動周波数を設定	
例）SPECIAL 16 30.0		
指 3 の振動数を設定するが，振動を ON/OFF する機能は無い．振動を ON/OFF には，SPECIAL 13 を実行する．		
	引数の種類	説明
引数 1	VAR	振動数［Hz］
引数 2	なし	

８-２-３ 表示に関する SPECIAL コマンド

番号	名称	
20	BAR グラフ表示を ON/OFF する.	
例）SPECIAL 20 0 1		
バーグラフを画面上に表示する. 表示位置，大きさなどの指定はできない．これら設定はメインプログラム(HH.CPP)で行う.		
	引数の種類	説明
引数１	VAR	BAR グラフの番号（０，１）
引数２	VAR	ＯＮ：１　ＯＦＦ：０

８-３ SPECIAL コマンドを使用したプログラムの例

```
4.0
AFC(Vibration): 90 度 -> CW/CCW : Robomec2k 用
//        VAR02 : tip_angle
//        VAR10 : KEY_IN
//        VAR11 : AFC_1
//        VAR12 : AFC_2
//        VAR13 : AFC_3
//        AGL01 : 初期姿勢
//        AGL02 : 初期姿勢
//        AGL03 : 初期姿勢

          VAR02 = 80.0      // tip_angle
          VAR11 = 0.0 // finger1 = off
          VAR12 = 0.0 // finger2 = off
          VAR13 = 0.0 // finger3 = off

//        AFC 周波数設定
          SPECIAL 14.0 30.0
          SPECIAL 15.0 40.0
          SPECIAL 16.0 50.0

// BAR_GRAPH1,2 on
          SPECIAL     20.0  0.0         1.0
          SPECIAL     20.0  1.0         1.0

          PTP_ANGLE   F1    -40.0 0.0 VAR02
          PTP_ANGLE   F2    -40.0 0.0 VAR02
          PTP_ANGLE   F3    -40.0 0.0 VAR02
          MESSAGE 対象物をセットする準備をして下さい.
          KEY_IN
          SILENT_ON
          SLEEP 1000.0
          REL_PTP_ANGLE     F1    0.0 0.0 0.0
          REL_PTP_ANGLE     F2    0.0 0.0 0.0
          REL_PTP_ANGLE     F3    0.0 0.0 0.0
          WAIT_INPOS  ALL
//        初期姿勢を記憶
          TEACH_ANGLE F1    AGL01
          TEACH_ANGLE F2    AGL02
          TEACH_ANGLE F3    AGL03
          SPECIAL     1
          MESSAGE キーを押すと指は離れます.
          FLUSH_BUF
FIRST:
          SPECIAL 2   2.0
          IF !KB_HIT  GOTO  FIRST:
          KEY_IN      VAR10
          IF VAR10 == 48 GOSUB AFC_OFF_ALL:
          IF VAR10 == 49 GOSUB TOGGLE_AFC1:
          IF VAR10 == 50 GOSUB TOGGLE_AFC2:
          IF VAR10 == 51 GOSUB TOGGLE_AFC3:
          IF VAR10 == 52 GOSUB AFC_ON_ALL:
          IF VAR10 == 32 GOTO FIRST_END:
          GOTO  FIRST:
FIRST_END:
          GOSUB AFC_OFF_ALL:

// BarGraph を消去
          SPECIAL     20.0 0.0 0.0
          SPECIAL 20.0 1.0 0.0
GRASP:
          GOSUB AFC_OFF_ALL:
          TORQUE      F1    60.0 60.0 60.0
          TORQUE      F2    60.0 60.0 60.0
          TORQUE      F3    60.0 60.0 60.0

          MESSAGE 把握したらキーを押してください.
          KEY_IN

// BAR_GRAPH1,2 off
          SPECIAL     20.0  0.0         0.0
          SPECIAL     20.0  1.0         0.0

          PTP_ANGLE   J12   10.0
          PTP_ANGLE   J13   85.0
          PTP_ANGLE   J22   10.0
          PTP_ANGLE   J23   90.0
          PTP_ANGLE   J32   10.0
          PTP_ANGLE   J33   90.0
          WAIT_INPOS  ALL
          TORQUE            J11   10.0
          TORQUE            J21   10.0
          TORQUE            J31   10.0
          MESSAGE 初期姿勢をとったらキーを押して下さい
          KEY_IN
          REL_PTP_ANGLE     F1    0.0 0.0 0.0
          REL_PTP_ANGLE     F2    0.0 0.0 0.0
          REL_PTP_ANGLE     F3    0.0 0.0 0.0
          WAIT_INPOS  ALL

//        初期姿勢を記憶
          TEACH_ANGLE F1    AGL01
          TEACH_ANGLE F2    AGL02
          TEACH_ANGLE F3    AGL03
```

```
//        AFC 周波数設定
          SPECIAL 15.0 60.0
          SPECIAL 16.0 60.0

          MESSAGE キーを押すと Rolling スタート
          KEY_IN
          FLUSH_BUF
          REL_PTP_ANGLE     F1    0.0 0.0 0.0
          REL_PTP_ANGLE     F2    0.0 0.0 -10.0
          REL_PTP_ANGLE     F3    0.0 0.0 -10.0
          WAIT_INPOS  ALL
          MESSAGE [1]=CW, [2]=CCW, [0]=STOP, [3]=QUIT
          FLUSH_BUF
LOOP:
          IF !KB_HIT GOTO LOOP:
          KEY_IN      VAR10
          FLUSH_BUF
          IF VAR10 == 48 GOSUB AFC_OFF_ALL:
          IF VAR10 == 49 GOSUB CW:
          IF VAR10 == 50 GOSUB CCW:
          IF VAR10 == 51 GOTO STOP:
          GOTO LOOP:
STOP:
          SILENT_OFF
          GOSUB AFC_OFF_ALL:
          EXIT

// CW 方向に対象物を回転
CW:
   // 指１に振幅 2.0[V]の振動を与える（モード４/CW）
          SPECIAL 11.0 4.0 2.0
          RETURN

CCW:
   // 指１に振幅 2.0[V]の振動を与える（モード６/CCW）
          SPECIAL 11.0 6.0 2.0
          RETURN

// finger1 の AFC 出力を反転
TOGGLE_AFC1:
          IF VAR11 > 0.0 GOTO AFC1_OFF:
          SPECIAL 11.0 1.0 2.0
          LAMP_ON     0.0
          VAR11 = 1.0
          RETURN
AFC1_OFF:
          SPECIAL 11.0 0.0 0.0
          LAMP_OFF 0.0
          VAR11 = 0.0
          RETURN

// finger2 の AFC 出力を反転
TOGGLE_AFC2:
          IF VAR12 > 0.0 GOTO AFC2_OFF:
          SPECIAL 12.0 1.0 2.0
          LAMP_ON     1.0
          VAR12 = 1.0
          RETURN
AFC2_OFF:
          SPECIAL 12.0 0.0 0.0
          LAMP_OFF 1.0
          VAR12 = 0.0
          RETURN

// finger3 の AFC 出力を反転
TOGGLE_AFC3:
          IF VAR13 > 0.0 GOTO AFC3_OFF:
          SPECIAL 13.0 1.0 2.0
          LAMP_ON     1.0
          VAR13 = 1.0
          RETURN
AFC3_OFF:
          SPECIAL 13.0 0.0 0.0
          LAMP_OFF 1.0
          VAR13 = 0.0
          RETURN

// 全ての finger の AFC 出力を OFF
AFC_OFF_ALL:
          SPECIAL 11.0 0.0 0.0
          SPECIAL 12.0 0.0 0.0
          SPECIAL 13.0 0.0 0.0
          LAMP_OFF    0.0
          LAMP_OFF    1.0
          VAR11 = 0.0
          VAR12 = 0.0
          VAR13 = 0.0
          RETURN

// 全ての finger の AFC 出力を ON
AFC_ON_ALL:
          SPECIAL 11.0 1.0 2.0
          SPECIAL 12.0 1.0 2.0
          SPECIAL 13.0 1.0 2.0
          LAMP_ON     0.0
          LAMP_ON     1.0
          VAR11 = 1.0
          VAR12 = 1.0
          VAR13 = 1.0
          RETURN
END
```

９　エラーメッセージ

以下にエラーメッセージの一覧を記す.

●エラーは全て非常停止扱いとする.

●エラー要因は EMG.log および画面上に出力される.

●メッセージは，「"TIP_ERR:"　エラーメッセージ　(種別)」の形式で出力される．種別は，"PreprocessError"，"SyntaxError"，"RuntimeError"の 3 種類とする.

９-１ 解析時エラー(PreprocessError)

Can not allocate memory for ～

メモリを動的に確保することができませんでした．TIP プログラムが大き過ぎる,ラベル数が多すぎる，使用する変数が多すぎる，などです．ちなみに REM コマンド行の数は記憶領域の大きさに関係ありません．MS-DOS の知識があるならば，使用可能領域を増やす方法もあります（不要なデバイスドライバを外す，UMA メモリを活用するなど）．あるいはログ記録用領域を小さくすることで使用可能メモリ領域を増やすことは可能です．一旦,ＰＣをリセットし，再起動すると良くなる可能性もあります.

File can not write(ファイル名)

ファイルに書き込むことができません．たとえばエディタなどで開いていて共有違反が発生した場合などです．開いているファイルを閉じて下さい.

File not fond(ファイル名)

読み込もうとしたファイルがカレントディレクトリに存在しません.

Too long filename

ファイル名（フォルダ名含む）が長すぎた場合に発生します．問題を解決するには，フォルダ名を短くする（ディレクトリを深くしない）か，ソースリストを変更（TIP_MAX_FNAME を大きくする）する必要がある.

Format error

以下にあげる書式エラーが存在します.

1. 一行の文字数が TIP_MAX_CHAR よりも長い.
2. ラベルの文字数が TIP_LABEL_MAX_CHAR（＝３2）よりも長い.

MESSAGE コマンドの文字列長が TIP_MAX_MESSAGE_LENGTH よりも長い,あるいは文字列中に改行(¥n)を含む場合.

Wrong equation format

数式の書式にミスがあります.

異なった型同士の演算や定数への代入が考えられます.

Wrong version number

バージョン番号が一致していません．バージョン番号が一致していない場合，コマンドの仕様が異なる恐れがあります．上位互換とも限りませんので最新仕様に合わせてスクリプトを変更するか,あるいは同一のバージョンの実行形式を用いて実行させてください.

９-２ 文法エラー(SyntaxError)

Duplicated label name

同じ名前のラベルが宣言されています．Loop 以外のラベルは 1 プログラム中に 1 箇所でしか宣言してはいけません．

Label ″ラベル名″ does not exist

GOTO で指定されたラベルが存在しません．もう一度確認してください．

Not exist END

プログラムが END 行で終了していない．

Unknown command

コマンド名が間違っています．予約語で定義されていないコマンドが使用されている．打ち間違いか，あるいはラベルのコロン（":"）を付け忘れた場合が考えられる．

Wrong argument

引数の指定が間違っています．コマンド毎に必要とする引数が異なります．また引数の型も異なります．リファレンスマニュアルを参照してください．

９-３ 実行時エラー(RuntimeError)

Domain error

演算子（たとえばＳＱＲＴ）の引数が不正な範囲（＜０）である．

Invalid Return command

対応する GOSUB/IF_GOSUB の存在しない RETURN コマンドを実行しようとした．

Possible use of Variable before definition

変数が値を代入される前に参照されている．

Unknown command

コマンド名が間違っています．予約語で定義されていないコマンドが使用されている．打ち間違いか，あるいはラベルのコロン（":"）を付け忘れた場合が考えられる．

Wrong argument

引数の指定が間違っています．コマンド毎に必要とする引数が異なります．また引数の型も異なります．リファレンスマニュアルを参照してください．

Wrong step number

GOTO 系のジャンプ先が不正なプログラムステップ番号である．プログラム解析時にエラーチェックするはずだが，それをスリ抜けてしまった場合を考慮．

以下に TIP インタープリタ設計用資料を記す．ただし，一部，古い資料もあるので，実際の tip.cpp/tip_if.cpp/tip.h/tip_if.h の内容とは一致しない個所もあります．

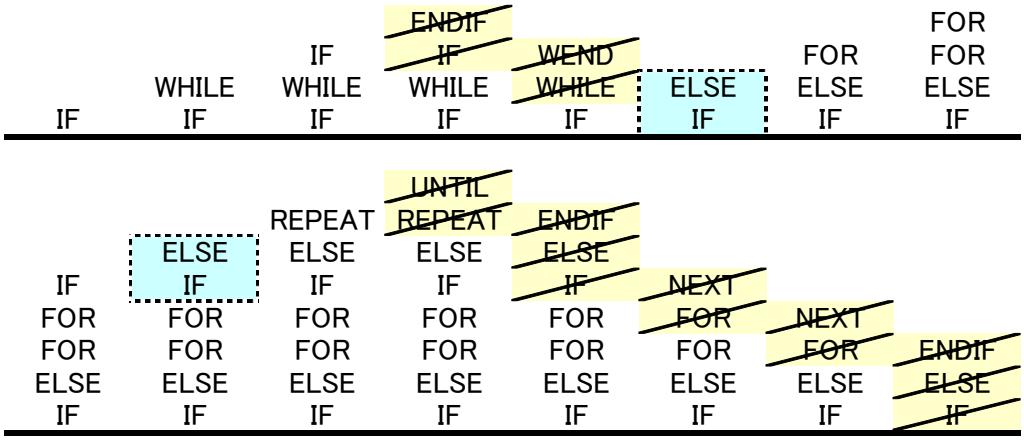
１０-１ 制御構文の構造解析

Ver.3.0 から下記の制御構文が導入されたことに伴い，Phase4 で構造のチェックを行う必要が出てきた．
制御構文の構造チェックはインタープリタによる解析時にのみ行われる．実行時には，あまりチェックを行わないので，スパゲッティー状のプログラムを作成した結果，予期しない動作をする可能性がある．しかし，これは TIP プログラマーの責任である．

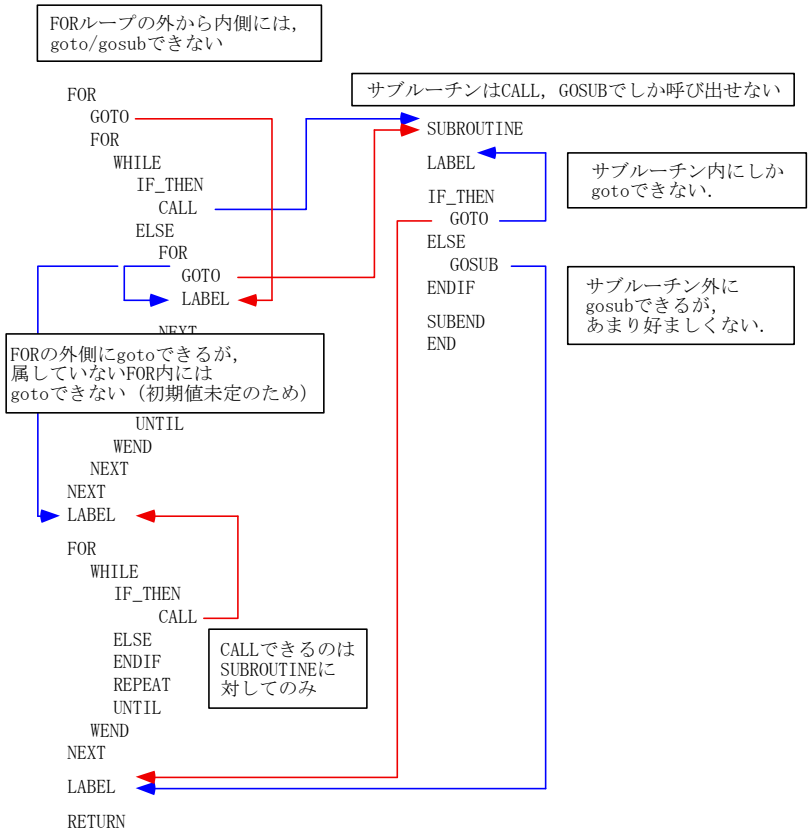
（制御構文）

1. IF-THEN-ELSE-ENDIF
2. WHILE-WEND
3. REPEAT-UNTIL
4. FOR-NEXT
5. SUBROUTINE-SUBEND/CALL

Step_no					
0010	IF				
0012		WHILE			
0015			IF		
0017			ENDIF		
0018		WEND			
0020	ELSE				
0022		FOR			
0024			FOR		
0025				IF	
0027				ELSE	
0029					REPEAT
0032					UNTIL
0033				ENDIF	
0034			NEXT		
0035		NEXT			
0036	ENDIF				



構文のチェックは，上記のような１対１対応のチェック以外にも，ジャンプ系コマンドのチェックも必要になる．以下の図は７-６．制御構文にある図と同じ物である．



ルールを要約すると以下のようになる．

（１） IF_THEN_ELSE_ENDIF，WHILE-WEND，REPEAT-UNTIL に関しては，どこから飛んできても，一向に構わない．したがって，これらに関してはチェックしない．

（２） FOR に関しては，自分の属している FOR 以外の FOR 内に goto，gosub してはいけない．なぜならば，その FOR のカウンタに用いられる汎用変数の初期値は未定である[13]．

（３） SUBROUTINE 内から外に goto は禁止だが，gosub は構わない．

（４） SUBROUTINE 外から内への goto は禁止だが，gosub は構わない．

（５） SUBROUTINE 内から内への goto，gosub は構わない．

[13] もし，初期値未定の問題を気にしないならば，この制約は外しても構わない．

１０-２　定数宣言

定数名	説明	初期値
TIP_MAX_CHAR	TIP プログラム 1 行の最大文字数	1023+1
TIP_FILENAME	TIP ファイル名の初期値	TITAN.TIP
TIP_FORMAL_FILENAME	中間言語ファイル名	FORMAL.TMP
TIP_LABEL_FILENAME	ラベルテーブルファイル名	LABEL.TMP
TIP_DEFAULT_VALUE_DOUBLE	各変数の初期値（未代入状態）	99999999.9
TIP_MAX_FNAME	ファイル名の最大文字列	1023+1
TIP_LABEL_MAX_CHAR	ラベル名の最大文字数	32
TIP_MAX_LEG	足の数	4
TIP_MAX_JOINT	１本脚あたりの最大関節数	3
TIP_MAX_MESSAGE_LENGTH	MESSAGE コマンドで表示可能な文字列の長さ	70 [byte]
TIP_A_MAX_SLOW_RATE	SLOW 指定時の割合-	50[%]
TIP_A_MAX_FAST_RATE	FAST 指定時の割合	150[%]
TIP_V_MAX_SLOW_RATE	SLOW 指定時の割合-	50[%]
TIP_V_MAX_FAST_RATE	FAST 指定時の割合	150[%]
TIP_OMEGA_MAX_SLOW_RATE	SLOW 指定時の割合-	50[%]
TIP_OMEGA_MAX_FAST_RATE	FAST 指定時の割合	150[%]
TIP_OMEGA_DOT_MAX_SLOW_RATE	SLOW 指定時の割合-	50[%]
TIP_OMEGA_DOT_MAX_FAST_RATE	FAST 指定時の割合	150[%]
TIP_TORQUE_MAX_SOFT_RATE	SOFT 指定時の割合	50[%]
TIP_TORQUE_MAX_FIRMLY_RATE	FIRMALY 指定時の割合	150[%]
TIP_A_MAX_RATE_MAXIMUM	RATE 指定時の最大値	200[%]
TIP_V_MAX_RATE_MAXIMUM	RATE 指定時の最大値	200[%]
TIP_MAX_SUBROUTINE_CALL	GOSUB によるサブルーチンコールの最大ネスト深さ	10

１０-３　構造体宣言

１０-３-１　メイン

```
struct tipT {
    char str[TIP_MAX_CHAR+1];              /* ファイルから読み込む文字列 */
     char tip_directory[TIP_MAX_FNAME - (8 + 3 + 1 + 1)]; // TIPディレクトリ（ソース，中間ファイル用）
    char tip_fname[TIP_MAX_FNAME];         /* TIPファイル名 */
    char formal_fname[TIP_MAX_FNAME]; /* 中間言語ファイル名 */
    char label_fname[TIP_MAX_FNAME];       /* ラベルテーブルファイル名 */
    FILE *fp_tip;
    FILE *fp_formal;
    FILE *fp_label;
    int step_no;                /* program step number */
    int max_step_no;                 /* プログラムステップの最大値 */
    struct tip_programT far          *program;
    struct tip_variable_tableT       variable;
    struct tip_label_tableT          label_table;
     struct tip_string_tableT        string_table;
     struct tip_return_step_stackT   return_step_stack;    /* GOSUB/RETURN */
     /* エラー関係              */
     enum tip_errT   err_code;
     char            err_message[TIP_MAX_MESSAGE_LENGTH+1];
     double          tmp;       /* 汎用の変数．主にpossible use～の変数の添え字の受け渡し  */
     /* バージョン番号          */
     double               version;   /* 呼び出し側で値をセットする    */
     /* その他                  */
     enum tip_phaseT phase;     /* いまどのフェーズ？(PAHT1, 2, 3, RUNNING)      */
};
```

１０-３-２　変数領域

```
/***** 変数領域の定義 *****/
/* POSnn 変数の型定義 */
struct tip_posT {
     double x, y, z;
};
/* AGLnn 変数の型定義 */
struct tip_aglT {
     double joint [TIP_MAX_JOINT];
};
/* TRQnn 変数の型定義 */
struct tip_trqT {
     double joint[TIP_MAX_JOINT];
};
/* 変数テーブルの定義 */
struct tip_variable_tableT {
     unsigned char *using;       /* 変数使用履歴テーブル */
     int max_val_number;         /* 使用されている VALnn の個数 */
     int max_pos_number;         /* 使用されている POSnn の個数 */
     int max_agl_number;         /* 使用されている AGLnn の個数 */
     int max_trq_number;         /* 使用されている TRQnn の個数 */
     int var_number[];           /* VARnn の添え字を管理      */
     dobule var_value[];              /* VARnn の値を管理          */
```

```
    int pos_number[];           /* POSnn の添え字を管理      */
    struct tip_posT pos_value[];    /* POSnn の値を管理          */
    int agl_number[];           /* AGLnn の添え字を管理      */
    struct tip_aglT agl_value[];    /* AGLnn の値を管理          */
    int trq_number[];           /* TRQnn の添え字を管理      */
    struct tip_trqT trq_value[];    /* TRQnn の値を管理          */
};
```

*using は，解析時に，calloc(sizeof(unsigned char), TIP_MAX_VARIABLE) で確保し，終わったら開放する．もし VARnn が使用されている場合は，using[nn] |= 1，POSnn が使用されている場合は，using[nn] |= 2，AGLnn が使用されている場合は，using[nn] |= 4，TRQnn が使用されている場合は，using[nn] |= 8.

*using を集計することで，各変数毎に使用されている変数の数が分かります．max_???_number は，その使用個数（ｎｎの最大値ではなく）を変数毎に保持します．この変数の値に従って添え字管理と値管理用の実際の配列を allocate します．

１０-３-３　ラベル変換テーブル

```
/***** ラベル変換テーブルの定義 *****/
struct tip_label_tableT {
    int max_label_number;/* 使用されているラベルの数 */
    char label_name[][32]; /* ラベル名を一時的に保持(解析時のみ必要) */
    int step_number[];          /* そのラベルの存在する step_number を保持する */
};
```

label_name[][32]，step_number[]は，Path2 でラベル数(max_label_number)が判明したら，一時的に allocate する．Path3 でこのテーブルを元にして，各 GOTO 部分に，直に step_number を埋め込んでしまうため，Path3 が終わったら free して構わない．

１０-３-４　文字列テーブル

```
/**** 文字列テーブル          ****/
struct tip_string_tableT {
   int   max_num;
   char *string;
   /* sizeof(char) * TIP_MAX_MESSAGE_LENGTH * max_num*/
};
```

MESSAGE コマンドは，このテーブルに格納された文字列を表示する．

１０-３-５　TIP プログラム格納用配列の構造

(a) TIP コマンド：Cmd

```
enum tip_commandT {
      NOP,
    PTP,
    REL_PTP,
    PTP_ANGLE,
    REL_PTP_ANGLE,
    ALL_STOP,
    ALL_STOP_RELEASE,
    INITIAL_POSTURE,
    STAND_UP,
    SIT_DOWN,
      VEL_CONTROL,
      OMEGA_CONTROL,
    SLEEP,
    WAIT_INPOS,
    WAIT_NEAR,
    COMPLIANCE,
    NO_COMPLIANCE,
    TORQUE,
    TEACH_POS,
    TEACH_ANGLE,
    TEACH_TORQUE,
    GOTO,
    IF_GOTO_EQ,
    IF_GOTO_NE,
    IF_GOTO_GT,
    IF_GOTO_LT,
    IF_GOTO_GE,
    IF_GOTO_LE,
    GOSUB,
    RETURN,
    IF_GOSUB_EQ,
    IF_GOSUB_NE,
    IF_GOSUB_GT,
    IF_GOSUB_LT,
    IF_GOSUB_GE,
    IF_GOSUB_LE,
      IF_GOTO_EVENT,
      IF_GOSUB_EVENT,
    LABEL_CMD,
    AD_IN,
    KEY_IN,
    MESSAGE,
    MESSAGE_CLEAR,
    OUTPUT_VARIABLES,
    FLUSH_BUF,
#if 0
    REM,
#endif
    EXIT,
    END,
    LOG_START,
    LOG_STOP,
    EMG_ON,
    EMG_OFF,
    EQUAL,
    EQUAL_ADD,
    EQUAL_SUB,
    EQUAL_MULT,
    EQUAL_DIV,
    EQUAL_ABS,
    EQUAL_INT,
    EQUAL_SQRT,
     EQUAL_SIN,
    EQUAL_COS,
    EQUAL_TAN,
    EQUAL_ASIN,
    EQUAL_ACOS,
    EQUAL_ATAN,
    EQUAL_ATAN2,
    EQUAL_DEG2RAD,
    EQUAL_RAD2DEG,
    EQUAL_NORM,
    EQUAL_UNIT_VECT,
    SET_A_MAX,
    SET_V_MAX,
    SET_OMEGA_MAX,
    SET_OMEGA_DOT_MAX,
    SET_TORQUE_MAX,
    SET_A_MAX_RATE,
    SET_V_MAX_RATE,
    SET_OMEGA_MAX_RATE,
    SET_OMEGA_DOT_MAX_RATE,
    SET_TORQUE_MAX_RATE,
    SET_A_MAX_SLOW,
    SET_A_MAX_FAST,
    SET_A_MAX_NORMAL,
    SET_V_MAX_SLOW,
    SET_V_MAX_FAST,
    SET_V_MAX_NORMAL,
    SET_OMEGA_MAX_SLOW,
    SET_OMEGA_MAX_FAST,
    SET_OMEGA_MAX_NORMAL,
    SET_OMEGA_DOT_MAX_SLOW,
    SET_OMEGA_DOT_MAX_FAST,
    SET_OMEGA_DOT_MAX_NORMAL,
    SET_TORQUE_MAX_FIRMLY,
    SET_TORQUE_MAX_SOFT,
    SET_TORQUE_MAX_NORMAL,
    SET_JUST_POSITION,
    SET_NEAR_POSITION,
    END_OF_CMD
};
```

(b) ターゲット：Target

```
enum tip_target_typeT {
    /* 脚／関節識別子 */
    NO_TARGET,
    TARGET_ALL,
    TARGET_F1, TARGET_F2, TARGET_F3, TARGET_F4, TARGET_F5, TARGET_F6, TARGET_F7, TARGET_F8, TARGET_F9,
    TARGET_J11, TARGET_J12, TARGET_J13, TARGET_J14, TARGET_J15, TARGET_J16, TARGET_J17, TARGET_J18, TARGET_J19,
    TARGET_J21, TARGET_J22, TARGET_J23, TARGET_J24, TARGET_J25, TARGET_J26, TARGET_J27, TARGET_J28, TARGET_J29,
    TARGET_J31, TARGET_J32, TARGET_J33, TARGET_J34, TARGET_J35, TARGET_J36, TARGET_J37, TARGET_J38, TARGET_J39,
    TARGET_J41, TARGET_J42, TARGET_J43, TARGET_J44, TARGET_J45, TARGET_J46, TARGET_J47, TARGET_J48, TARGET_J49,
    TARGET_J51, TARGET_J52, TARGET_J53, TARGET_J54, TARGET_J55, TARGET_J56, TARGET_J57, TARGET_J58, TARGET_J59,
    TARGET_J61, TARGET_J62, TARGET_J63, TARGET_J64, TARGET_J65, TARGET_J66, TARGET_J67, TARGET_J68, TARGET_J69,
    TARGET_J71, TARGET_J72, TARGET_J73, TARGET_J74, TARGET_J75, TARGET_J76, TARGET_J77, TARGET_J78, TARGET_J79,
    TARGET_J81, TARGET_J82, TARGET_J83, TARGET_J84, TARGET_J85, TARGET_J86, TARGET_J87, TARGET_J88, TARGET_J89,
    TARGET_J91, TARGET_J92, TARGET_J93, TARGET_J94, TARGET_J95, TARGET_J96, TARGET_J97, TARGET_J98, TARGET_J99,
    /* I/O識別子 */
    TARGET_VR1, TARGET_VR2, TARGET_VR3,
    TARGET_PAW1, TARGET_PAW2, TARGET_PAW3, TARGET_PAW4,
    TARGET_ANTENNA,
      /* 算術演算 */
      CAL_LEFT_ADD, CAL_LEFT_SUB, CAL_LEFT_MULT, CAL_LEFT_DIV,
      END_OF_TARGET
};
```

(c) 引数の種類：Arg_type

```
enum tip_argument_typeT {
    ARG_NONE,
    LABEL,
    /* EVENT */
    NO_EVENT,
    HALT_REQUEST,
      KB_HIT,
      INPOS_F1, INPOS_F2, INPOS_F3, INPOS_F4,
INPOS_ALL,
      NEAR_F1, NEAR_F2, NEAR_F3, NEAR_F4, NEAR_ALL,
      /**** 文字列(テーブル番号) ****/
      STRING,
      /**** 定数 ****/
    CONSTANT,
    REL_CONSTANT,
    /**** 変数(正数) ****/
    VAR,
      /* 構造体変数    */
    POS, AGL, TRQ,
      /* 座標変数           */
    POS_X, POS_Y, POS_Z,
    /* 角度・トルク */
      AGL_1, AGL_2, AGL_3,
      AGL_4, AGL_5, AGL_6,
      AGL_7, AGL_8, AGL_9,
      TRQ_1, TRQ_2, TRQ_3,
      TRQ_4, TRQ_5, TRQ_6,
      TRQ_7, TRQ_8, TRQ_9,
    /* 相対位置 */
    REL_VAR,
    REL_POS, REL_AGL, REL_TRQ,
    REL_POS_X, REL_POS_Y, REL_POS_Z,
    REL_AGL_1, REL_AGL_2, REL_AGL_3,
    REL_AGL_4, REL_AGL_5, REL_AGL_6,
    REL_AGL_7, REL_AGL_8, REL_AGL_9,
    REL_TRQ_1, REL_TRQ_2, REL_TRQ_3,
    REL_TRQ_4, REL_TRQ_5, REL_TRQ_6,
    REL_TRQ_7, REL_TRQ_8, REL_TRQ_9,

    /**** 変数(反転) ****/
    M_VAR,
      /* 構造体変数    */
    M_POS, M_AGL, M_TRQ,
      /* 座標変数           */
    M_POS_X, M_POS_Y, M_POS_Z,
    /* 角度・トルク */
      M_AGL_1, M_AGL_2, M_AGL_3,
      M_AGL_4, M_AGL_5, M_AGL_6,
      M_AGL_7, M_AGL_8, M_AGL_9,
      M_TRQ_1, M_TRQ_2, M_TRQ_3,
      M_TRQ_4, M_TRQ_5, M_TRQ_6,
      M_TRQ_7, M_TRQ_8, M_TRQ_9,
    /* 相対位置 */
    M_REL_VAR,
    M_REL_POS,    M_REL_AGL,    M_REL_TRQ,
    M_REL_POS_X, M_REL_POS_Y, M_REL_POS_Z,
    M_REL_AGL_1, M_REL_AGL_2, M_REL_AGL_3,
    M_REL_AGL_4, M_REL_AGL_5, M_REL_AGL_6,
    M_REL_AGL_7, M_REL_AGL_8, M_REL_AGL_9,
    M_REL_TRQ_1, M_REL_TRQ_2, M_REL_TRQ_3,
    M_REL_TRQ_4, M_REL_TRQ_5, M_REL_TRQ_6,
    M_REL_TRQ_7, M_REL_TRQ_8, M_REL_TRQ_9,
    END_OF_ARG
};
```

TIP プログラムの格納用配列の構造には，もう一工夫必要かも知れない．ちょっと効率の悪い方法を選択している．ただ，コマンド毎に引数の数が違うからといって各個，allocate するのは確保するのも面倒ならば,開放するのも面倒．果たして効率良くメモリ管理されるのかどうかも不明なので,かなり冗長な方法を選択した．ただし MESSAGE 命令だけは，文字列を引数として持たねばならないので，MESSAGE 領域をテーブルとして確保せざるを得ない

１０-３-６　GOSUB/RETURN 用スタック

```
struct tip_return_step_stackT {
    int     return_step[TIP_MAX_SUBROUTINE_CALL];   /* < 0 ならばエラー      */
    int     nest;                                   /* 現在の深さ．初期値０          */
};
```

リスト構造によるスタックを構築すると，GOSUB/RETURN のネストはメモリの許す範囲内で無制限にコールすることが可能である．できあがったコードもエレガントに見えるのだが，実際にはネストの深さに制限を設ける本データ構造のメリットの方が多い．そもそも GOSUB/RETURN のネストが無制限である必然性があるだろうか？　無制限に GOSUB が呼ばれる状況のほとんどは，TIP プログラムのアルゴリズムにミスがあった場合，すなわち無限ループに陥った場合に他ならない．そのような場合には，早めにエラーを発するべきだろう．

step_no	TIP プログラム	
0020	GOSUB L1:	(1)
0021	STAND_UP 100	
0040	L1:	
0041	RETURN	(2)

nest =
 tip.return_step_stack.nest
return_step[] ＝　tip.return_step_stack.return_step[]
step_no =
　tip.step_no

［初期状態］
　nest == 0,　return_step[]={-1,...,-1}

［GOSUB(1)］
・nest==0 の場合，
　return_step[0] = 0021,　nest++,　step_no++

［RETURN(2)］
・nest==1 の場合，
　nest--,　step_no = return_step[0] = 0021,　return_step[0] = -1

帰納的にコードとして表現すると，

［GOSUB 時］
もし nest < TIP_MAX_SUBROUTINE_CALL ならば，
・çreturn_step[nest] = step_no + 1
・çnest ++
・çstep_no ++

［RETURN 時］
もし nest > 0 ならば，
・nest --
・step_no = return_step[nest]
・return_step[nest] = -1

１０-４ 各種ファイルの書式

１０-４-１ プログラム例

以下のプログラム例は Ver.3.0 以前のものなので，Ver.3.0 以降では正常に動きません．近日中に Ver.3.0 対応のサンプルプログラムに差し替えます.

```
2.0
４脚一気に動かすタイプの胴体接地型歩容

        VAR01 = 70.0
        VAR02 = 301.0 + 90.0
        VAR03 = 50.0                    // 脚を上げる高さ
        ptp F1 -VAR02  101.0 -200.0     // AT_P1の初期姿勢の値を目標値
        ptp F2  VAR02  101.0 -200.0
        ptp F3 -VAR02 -301.0 -200.0
        ptp F4  VAR02 -301.0 -200.0
        VAR04 = 10.0                    // VR1のゲイン
        VAR05 = 20.0                    // VR2のゲイン
        SET_A_MAX   ALL     1000.0
        SET_V_MAX   ALL      500.0

VR1_ORG:
        wait_inpos ALL
        AD_IN VR1    VAR06              // VR1の原点をリセット
VR2_ORG :
        wait_inpos ALL
        AD_IN VR2    VAR07              // VR2の原点をリセット
VR1_CHECK :
        wait_inpos ALL
        AD_IN VR1 VAR10
        VAR10 = VAR10 - VAR06
        VAR10 = VAR10 * VAR04
        if VAR10 <=  100.0 GOTO VR1_CHECK1:
        VAR10 =  100.0
VR1_CHECK1:
        if VAR10 > -100.0 GOTO VR1_CHECK2:
        VAR10 = -100.0
VR1_CHECK2:
        VAR100 = ABS VAR10
        if VAR100 < VAR01               GOTO    VR2_CHECK:
        GOTO START:
VR2_CHECK :
        wait_inpos ALL
        AD_IN VR2    VAR11
        VAR11 = VAR11 - VAR07
        VAR11 = VAR11 * VAR05
        VAR11 = -200.0 + VAR11
        ptp F1 -VAR02  101.0 VAR11
        ptp F2  VAR02  101.0 VAR11
        ptp F3 -VAR02 -301.0 VAR11
        ptp F4  VAR02 -301.0 VAR11
        GOTO VR1_CHECK:
START :
        ptp F1 R 0.0 R 0.0 R VAR03
        ptp F2 R 0.0 R 0.0 R VAR03
        ptp F3 R 0.0 R 0.0 R VAR03
        ptp F4 R 0.0 R 0.0 R VAR03
        wait_inpos ALL
        ptp F1 R 0.0 R 200.0 R 0.0          // F1を前方に10cm動かす
        ptp F2 R 0.0 R 200.0 R 0.0          // F2を前方に10cm動かす
        ptp F3 R 0.0 R 200.0 R 0.0          // F3を前方に10cm動かす
        ptp F4 R 0.0 R 200.0 R 0.0          // F4を前方に10cm動かす
        wait_inpos ALL
        ptp F1 R 0.0 R 0.0 R -VAR03         // F1を下方に5cm動かす
        ptp F2 R 0.0 R 0.0 R -VAR03         // F2を下方に5cm動かす
        ptp F3 R 0.0 R 0.0 R -VAR03         // F3を下方に5cm動かす
        ptp F4 R 0.0 R 0.0 R -VAR03         // F4を下方に5cm動かす
        wait_inpos ALL
        ptp F1 R 0.0 R 0.0 R -VAR03         // BODYを上方に5cm動かす
        ptp F2 R 0.0 R 0.0 R -VAR03
        ptp F3 R 0.0 R 0.0 R -VAR03
        ptp F4 R 0.0 R 0.0 R -VAR03
        wait_inpos ALL
        ptp F1 R 0.0 R -200.0 R 0.0         // BODYを前方に10cm動かす
        ptp F2 R 0.0 R -200.0 R 0.0
        ptp F3 R 0.0 R -200.0 R 0.0
        ptp F4 R 0.0 R -200.0 R 0.0
        wait_inpos ALL
        ptp F1 -VAR02  101.0 VAR11          // BODYを下方に5cm動かす
        ptp F2  VAR02  101.0 VAR11
        ptp F3 -VAR02 -301.0 VAR11
        ptp F4  VAR02 -301.0 VAR11
        wait_inpos ALL
        IF HALT_REQUEST             GOTO FINISH:
        GOTO VR2_ORG:
FINISH :
        initial_posture
        wait_inpos all

END
```

１０-４-２　FORMAL.TMP

余分な空白は取り除かれ，１行目からステップＮｏ．＝０で記述されている．基本的に各オペランド／オペコード間はタブ(¥t)で区切られている．Version 行および動作に無関係な REM 行も削除されている．

＜ステップＮｏ．＞　ラベル名：
＜ステップＮｏ．＞　＜コマンド名＞　＜引数１＞　＜引数２＞．．．
＜ステップＮｏ．＞　＜変数＞　＝　＜引数１＞　｛＜演算子＞　＜引数 2＞｝

例外：

＜ステップＮｏ．＞　ＭＥＳＳＡＧＥ　＜タブ＞　＜オリジナルの文字列（空白ＯＫ）＞

```
VAR01   = 70.0
VAR02   = 301.0         +       90.0
VAR03   = 50.0
ptp     F1-VAR02        101.0   -200.0
ptp     F2VAR02 101.0   -200.0
ptp     F3-VAR02        -301.0  -200.0
ptp     F4VAR02 -301.0  -200.0
VAR04   = 10.0
VAR05   = 20.0
SET_A_MAX ALL   1000.0
SET_V_MAX ALL   500.0
VR1_ORG:
wait_inposALL
AD_IN   VR1     VAR06
VR2_ORG:
wait_inposALL
AD_IN   VR2     VAR07
VR1_CHECK:
wait_inposALL
AD_IN   VR1     VAR10
VAR10   = VAR10 -       VAR06
VAR10   = VAR10 *       VAR04
if      VAR10   <=      100.0   GOTO    VR1_CHECK1:
VAR10   = 100.0
VR1_CHECK1:
if      VAR10   >       -100.0  GOTO    VR1_CHECK2:
VAR10   = -100.0
VR1_CHECK2:
VAR100  = ABS   VAR10
if      VAR100  <       VAR01   GOTO    VR2_CHECK:
GOTO    START:
VR2_CHECK:
wait_inposALL
AD_IN   VR2     VAR11
VAR11   = VAR11 -       VAR07
VAR11   = VAR11 *       VAR05
VAR11   = -200.0        +       VAR11
ptp     F1-VAR02        101.0   VAR11
ptp     F2VAR02 101.0   VAR11
ptp     F3-VAR02        -301.0  VAR11
ptp     F4VAR02 -301.0  VAR11
GOTO    VR1_CHECK:
START:
ptp     F1R     0.0     R       0.0     R       VAR03
ptp     F2R     0.0     R       0.0     R       VAR03
ptp     F3R     0.0     R       0.0     R       VAR03
ptp     F4R     0.0     R       0.0     R       VAR03
wait_inposALL
ptp     F1R     0.0     R       200.0   R       0.0
ptp     F2R     0.0     R       200.0   R       0.0
ptp     F3R     0.0     R       200.0   R       0.0
ptp     F4R     0.0     R       200.0   R       0.0
wait_inposALL
ptp     F1R     0.0     R       0.0     R       -VAR03
ptp     F2R     0.0     R       0.0     R       -VAR03
ptp     F3R     0.0     R       0.0     R       -VAR03
ptp     F4R     0.0     R       0.0     R       -VAR03
wait_inposALL
ptp     F1R     0.0     R       0.0     R       -VAR03
ptp     F2R     0.0     R       0.0     R       -VAR03
ptp     F3R     0.0     R       0.0     R       -VAR03
ptp     F4R     0.0     R       0.0     R       -VAR03
wait_inposALL
ptp     F1R     0.0     R       -200.0  R       0.0
ptp     F2R     0.0     R       -200.0  R       0.0
ptp     F3R     0.0     R       -200.0  R       0.0
ptp     F4R     0.0     R       -200.0  R       0.0
wait_inposALL
ptp     F1-VAR02        101.0   VAR11
ptp     F2VAR02 101.0   VAR11
ptp     F3-VAR02        -301.0  VAR11
ptp     F4VAR02 -301.0  VAR11
wait_inposALL
IF      HALT_REQUEST    GOTO    FINISH:
GOTO    VR2_ORG:
FINISH:
initial_posture
wait_inposall
END
```

１０-４-３　LABEL.TMP

＜種別＞　＜ラベル名＞　　＜ステップＮｏ．＞

```
L       VR1_ORG:        11
L       VR2_ORG:        14
L       LVR1_CHECK:     17
S       VR1_CHECK1:     24
S       VR1_CHECK2:     27
S       VR2_CHECK:31
L       START:          42
L       FINISH:         75
```

種別は,
　Ｌ：通常のラベル行
　Ｓ：サブルーチンのラベル行

１１　デバッグ方法

(a) ＴＩＰプログラムのデバッグ方法

TIP プログラムを実行すると，FORMAL.TMP が自動的に生成される．FORMAL.TMP は，ソースリスト中の不要なコードを削除し，フォーマットに則った引数のみを受け付けて切り出されたファイルである．FORMAL.TMP とソースリストを比較することで，大きなミスが発見される可能性も高い．また，Path1，Path2，Path3 の３段階の処理を経て，ソース(TIP)をメモリ内に格納するが，この過程で発見されたエラーは，tip->err_code に代入される．set_error_message()で文字列化されたエラー情報を見る（表示するように実装）ことで，エラーの種類と発生個所を知ることもできる．この時，エラーの発生した行番号は，FORMAL.TMP の行番号である．行番号は０から始まることにも注意すること．

(b) TIP インタープリターのデバッグ方法

TIP プログラマではなく，TIP を各ロボット制御プログラムに実装した場合に発生する様々なトラブルに対処するための手段として，tip.h 中の

```
#if 0
#define TIP_DEBUG
#endif
```

を

```
#if 1
#define TIP_DEBUG
#endif
```

とすることで，TIP.OBJ ファイルを生成することができる．TIP.OBJ ファイルは，Path3 終了後に生成される．中身は，メモリに格納された（バイナリ形式の）TIP コードをテキスト化したものである．以下のような形式である．

```
[0Line]                    ← FORMAL.TMP の行番号
Cmd        = EQUAL         ← コマンド名
Target     = NO_TARGET     ← ターゲット名
ArgType 1 = VAR            ← ３つのパラメータの型と値
ArgType 2 = CONSTANT
ArgType 3 = ARG_NONE
ArgValue1 = 1.000000
ArgValue2 = 30.000000
ArgValue3 = 0.000000
```

TIP インタープリターのコーディングにミスがあるか，このファイルを本仕様書と比較することで確認できる．

１２ 最後に

姿勢角・加速度センサに対しては，未だ使い道・使い方がはっきりとしていないので，割愛.

ただし，今後の TitanVIII の改造計画も考慮に入れると，

・汎用ＤＩＯ

・特別なＤＩＯを利用した高度なコマンド（FLOOR_DETECT の類）

が必要になると思われる．また，プログラムの構造化への対応，自由な変数名（個人的には反対派）への対応，なども考えられる.

１３ 謝辞

まずは，本体とモータドライバしか存在しなかった TitanVIII を電気的に動かすことが可能な状態まで完成させるために，昼夜を忘れて作業に没頭してくれた井上 智弘君（平成１０年度卒業）に感謝します．電源投入一発でミス無く動いた完成度の高さは特筆に価します．まだ，ＴＩＰが存在しない頃に，AUTO プログラム用の静歩行プログラムを一晩で作成，完成させたセンスも素晴らしい.

平成１１年度から包み込み歩容の研究プラットホームとして TitanVIII を本格的に使用し始めた，黒田則明氏，水野彰彦君にも感謝します．黒田氏には，巨大な木製の斜面製作や TitanVIII 足裏センサの設計・製作，ポテンショメータの交換など，ハードウェア面での強力なサポートを得ることができました．水野君は，足先接地歩容から包み込み歩容への遷移動作を実現するために，非常に少ない助言だけでＴＩＰを用いた本格的なプログラムを作成することで，ＴＩＰ実装の効果を実感させてくれました．両名共にＴＩＰプログラムの作成を通して数多くのバグを発見してくれました．また，新しいコマンドの追加は，主に両名からの指摘によるものであることを付記します．新しく追加されたコマンドを用いることで，より柔軟なプログラミングが可能になったと確信しています.