SeiralPort.h

このファイルでは、シリアルポートを介した通信を実現するための基本的な機能を実装した SerailCom クラスを定義している。

class SerialCom:

【クラスの概要】

このクラスは、シリアルポートで通信をするためのクラスである。通信パラメータは、データ長：8bit、パリティ：None、ストップビット：1 と固定されている。通信速度と通信モード（同期／非同期）のみ設定可能になっており、デフォルトでは、通信速度：115200 bps, 通信モード：同期型となっている。

多くのシリアルポートを使ったロボット制御では、同期型が使われているので、非同期モードの通信については、十分にテストされていないことに注意してください。

【メンバ変数】

Public 型：

char *device;

シリアルポートでバイス名。多くの UNIX 系のシステムでは、/dev/ttyS0 であり、Windows では、COM1 のような文字列になる。

int　baudrate;

シリアルポートの通信速度。B0, B9600, B115200 など、動作する OS によって、システムヘッダーに定義されている。Window の場合には、この定義がないので、SerialPort.h に定義している。

int mode;

シリアルポートの通信モード。mode=1 の場合は、非同期入出力モードとなる。通常は、同期モードで実施する方がよい。この実装では、非同期入出力モードに関して十分なテストがなされていない。

HANDLE handle;

シリアルポートの通信ハンドラ。UNIX 系のシステムでは、ファイルディスクプリタを表す int 型になる。

【コンストラクタとデストラクタ】

コンストラクタ：コンストラクタは、２つ用意されている。引数なしのコンストラクタでは、シリアルでバイスが NULL に設定されてしまうため、通信ポートを開く前にメンバ関

数 setDevPort を用いてシリアルポート名の設定する必要がある。

- SerialCom();
- SerialCom(char *devname);

デストラクタ：デストラクタでは、通信ポートのクローズを行い、シリアルポート名の削除を行う。

- ~SerialCom();

【メンバ関数】

Public 型：

void setDevPort(char*devname);

シリアルポート名の設定。引数で与えられた文字列を複製し、メンバ変数 device に代入する。

int openPort();

シリアルポートをオープンし、通信可能な状態にする。オープンするデバイス名は、メンバ変数 device で設定されたものである。シリアルポートのオープンに失敗した場合には、-1 を返す。成功した場合には、メンバ変数 handle を整数にキャストして返す。

void closePort();

シリアルポートをクローズし、メンバ変数 handle に H_NULL を代入する。

int Read(char *data, intlen);

この関数は、シリアルポート通信の低レベルのデータ受信関数である。シリアルポートから長さ len のデータを読み込み、そのデータを data の格納して返す。この関数の返り値は、読み込んだデータのバイト数になる。読み込みに失敗した場合には、-1 を返す。この関数は、UNIX 系の OS の read 関数と同等である。

int Write(char *data, intlen);

この関数は、シリアルポート通信の低レベルのデータ送信関数である。シリアルポートへ長さ len のデータ data を書き込む。この関数の返り値は、書き込んだデータのバイト数になる。読み込みに失敗した場合には、-1 を返す。この関数は、UNIX 系の OS の write 関数と同等である。

int recieveData(char *data, intlen);

この関数は、シリアルポートの高レベルデータ受信関数である。シリアルポートから len バイトデータを読み込み、data に格納する。返り値は、受信したデータのバイト数であり、失敗すれば-1 を返す。この関数は、データを受信する前に、メンバ関数 checkBuffer により、受信データがあるかどうかを確認する。もし、受信バッファに len バイトの受信データはがない場合には、１m 秒のスリープ状態に入り、100 回連続で受信バッファのデータが len バイトを満たさない時には、受信バッファの内容を削除し、－1 を返す。

受信バッファに len バイト以上データがあれば、len バイト受信バッファからデータを読み込み、data に格納してそのデータのバイト数を返す／

int sendData(char *data, intlen);

この関数は、シリアルポートの高レベルデータ送信関数である。引数 data の内容を len バイトだけシリアルポートに書き込み、書き込んだデータのバイト数を返す。何らかの理由で書き込みに失敗した場合（メンバ関数 Write が-1 を返したとき）エラーメッセージを表示し、－1 を返す。

int chkBuffer();

この関数は､シリアルポートの受信バッファにあるデータの長さを調べるための関数である。返り値は、受信バッファ内にあるデータのバイト数である。

int clearBuffer();

この関数は、シリアルポートの受信バッファの内容をすべて消去する。返り値は、削除したデータのバイト数であり、失敗すれば－1 を返す。

void printPacket(char *data, intlen);

標準エラー出力に、data の内容を len バイトだけ出力する。出力は、16 進数形式で表示される。このメンバ関数は、sendData 関数等で失敗した時に呼ばれている。

int getBaudrate();

この関数は、シリアルポートの通信速度を返す。実装では、メンバ変数 baudrate を返しているだけである。

int setBaudrate(int rate);

このメンバ関数は、シリアルポートの通信速度を rate に変更する。すなわちメンバ変数 baudrate に rate を代入し、その亜多雨を返す。

シリアルポートが既に接続されており、通信状態であればその通信速度を変更する。

int getMode();

この関数は、シリアルポートの通信モードを表すメンバ変数 mode の値を返す。

int setMode(int m);

この関数は、シリアルポートが通信状態でないときには、シリアルポートの通信モードを表すメンバ変数 mode に引数 m の値を代入し、その値を返す。シリアルポートが既に通信状態であれば、エラーメッセージを標準エラー出力に出力し、現在のメンバ変数 mode の値を返す。

int isConnected();

この関数は、シリアルポートが通信状態であるかどうかを確認する。通信状態であれば、1 をそうでなければ -1 を返す。

SerialRobot.h

このファイルでは、シリアルポートを介して動作制御を行うロボットの基本的な機能を提供するクラスを定義している。定義したクラスは、下記の３つのクラスである。

- RobotPosture クラス：ロボットの姿勢（各関節角の列）を表すクラスである。
- RobotMotion クラス:ロボットの動作を表すクラスであり、RobotPosture の列を表す。
- SerialRobot クラス：シリアルポートを用いた基本的な動作制御を行うためのクラス。

C 言語の関数:

short rad2deg(double d);

ラジアン→度への単位変換。　d*180/PI の結果を返す。

double deg2rad(short d);

度→ラジアンの単位変換。　d*PI/180 の結果を返す。

THREAD_FUNC thread_execution(void *args);

SerialRobot クラスのスレッドのメイン関数。SerialRobot のインスタンスを引数として、

メンバ関数 svc を呼び出すループを実行する。

C++のクラス：

class RobotPosture

【クラスの概要】

ロボットの姿勢を表すクラス。ロボットの各関節の角度およびその角度へ移動するまでの時間を保持する。

【メンバ変数】

Public 型：

int numJoints;

この変数は、関節の数を表す。

int *jointAngles;

この変数は、関節角の配列を表し、配列の長さは numJoints になっている。格納されるデータの単位は、度である。

double *jointAnglesRad;

この変数は、関節角の配列を表し、配列の長さは numJoints になっている。格納されるデータの単位は、ラジアンである。

double motionTime;

この変数は、jointAngles の姿勢までに移動するための時間を表している。単位は、ミリ秒とする。

【コンストラクタとデストラクタ】

コンストラクタ：コンストラクタは、ロボットの関節角数 n を引数とする。コンストラクタでは、関節角を保持するためのメンバ変数の配列 jointAngles, jointAnglesRad の領域確保と numJoints の設定を行う。また、motionTime は、100 に設定されるため、必要に応じてインスタンス生成後に設定する必要がある。

- RobotPosture(int n);

デストラクタ：デストラクタでは、メンバ変数の配列 jointAngles, jointAnglesRad の解放を行う。

- ~RobotPosture();

【メンバ関数】

Public 型：

RobotPosture *dupPosture();

新たに RobotPosture のインスタンスを生成し、メンバ変数 motionTime, jointAngles, jointAnglesRad をコピーして、そのインスタンスを返す**。すなわち自身の複製を生成する。**

double setMotionTime(double n);

メンバ変数の配列 jointAngles へ移動する時間を表すメンバ変数 motionTime に n を代入し、その値を返す。すなわち、motionTime=n;が実行される。

int getDegree(int id);

引数 id で指定された度数表現の関節角の値を返す。ただし、id は、1 から numJoints の間の整数である。すなわち、jointAngles[id -1]を返す。

double getRad(int id);

引数 id で指定されたラジアン表現の関節角の値を返す。ただし、id は、1 から numJoints の間の整数である。すなわち、jointAnglesRad[id -1]を返す

void setDegree(int id, intdeg);

引数 id で指定された度数表現の関節角を引数 deg に設定する。ただし、id は、1 から numJoints の間の整数である。すなわち、jointAngles[id -1]　= deg を実行する。

void setRad(int id, double rad);

引数 id で指定された度数表現の関節角を引数 rad に設定する。ただし、id は、1 から numJoints の間の整数である。すなわち、jointAnglesRad[id -1]　= rad を実行する。

void printPosture();

メンバ変数 jointAngles の内容を標準エラー出力に出力する。

void printPosture(std::ofstream& o);

メンバ変数 jointAngles の内容をファイルストリーム o に出力する。

void copyPosture(RobotPosture *p);

引数 p で与えられた RobotPosture に、メンバ変数 motionTime, jointAngles, jointAnglesRad の内容をコピーする。もし、p の関節角数が、自身の関節角数よりも少ない場合には、jointAngles, jointAnglesRad にコピーされる関節角は、0 から p->numJoints-1 までの要素である。ここで注意するのは、p の関節角数は変更されないということである。

void copyPosture(RobotPosture *p, int offset);

引数 p で与えられた RobotPosture に、メンバ変数 motionTime, jointAngles, jointAnglesRad の内容をコピーする。このとき、jointAngles, jointAnglesRad にコピーされる関節角は、引数 offset から始まる要素である。ここで注意するのは、p の関節角数は変更されないということである。

int *getJointAngles();

この関数は、内部変数である jointAngles を返す。

double *getJointAnglesRad();

この関数は、内部変数である jointAngles をラジアン表現に変換し、内部変数である jointAnglesRad へ代入しそれを返す。

bool equalTo(RobotPosture *pos);

この関数は、引数 pos で与えられた関節角列である jointAngles と自身の内部変数である jointAngles の各要素を比較して、すべてが一致している場合に、true を返し、それ以外であれば false を返す。

bool nearTo(RobotPosture *pos, int delta);

この関数は、引数 pos で与えられた関節角列である jointAngles と自身の内部変数である jointAngles の各要素を比較して、その差が delta より小さい場合に、true を返し、それ以外であれば false を返す。

class RobotMotion:

【クラスの概要】

このクラスは、RobotPosture の列として表現されたロボットの動作を規定するために定義されている。メンバ変数として、vector 型の motion、ロボットの関節角数である numJoints、現時点の RobotPosure を表す current、動作の評価順を規定する reverse をもつ。このクラスのインスタンスを用いて、ロボットの動作を定義する。

【メンバ変数】

Public 型:

std::vector<RobotPosture *> motion;

この変数は、ロボットの動作を表す。RobotPosture を要素に持つ vector 型で定義冴えている。

int　numJoints;

この変数は、ロボットの関節角数を表す。

int current;

この変数は、ロボットの姿勢列の現在の位置を表す。この変数は、メンバ関数 next を実行するときのインデックスとなる。

bool reverse;

この変数は、ロボットの動作を実行する場合に、motion の評価をする場合の順序を規定する。この変数が真であれば、メンバ関数 next で逆順にインデックスが進む。

【コンストラクタとデストラクタ】

コンストラクタ：コンストラクタでは、メンバ変数の初期化を行う。このコンストラクタでは、関節角数は 0 に設定されるため、メンバ関数 setJoints を用いてロボットの関節角数を指定しなければならない。

- RobotMotion();

デストラクタ：メンバ変数 motion の要素をすべて削除する。

- ~RobotMotion();

【メンバ関数】

Public 型：

int setJoints(int n);

メンバ変数 numJoints に n を代入し、その値を返す。

int getSize();

メンバ変数 motion のサイズを返す。この値は、ロボットの動作のキーポーズの数に一致している。

int setReverse(bool f);

メンバ変数 reverse に f を代入し、その値を返す。

int reset();

登録されている動作のキーポーズの位置（インデックス）を表すメンバ変数 current をリセットする。メンバ変数 reverse が偽の場合には、current=0　となり、真の場合には、キーポーズの最後の要素を指すインデックス、すなわち、current = motion.size() -1　に設定される。

RobotPosture *next();

登録された動作の中の current で表されたインデックスの要素である姿勢を返し、インデックス current を 1 つ進める。メンバ変数 reverse が真の場合には、逆方向になるため current を 1 つ減じる。

RobotPosture *get(int nth);

登録された動作の中の先頭から nth 番目の要素である姿勢を返す。

RobotPosture *rget(int nth);

登録された動作の中の末尾から nth 番目の要素である姿勢を返す。すなわち、この関数は、get(motion.size() – nth)　と同値である。

bool appendPosture(RobotPosture *js);

メンバ変数 motion の末尾に js を追加し、ture を返す。

bool insertPosture(RobotPosture *js, int nth);

メンバ変数 motion の先頭から nth 番目に js を追加し、ture を返す。

void deletePosture(int n);

メンバ変数 motion の先頭から nth 番目の要素を削除する。

bool loadMotionFromFile(const char *name);

name というファイルからロボットの動作データを読み込み、メンバ変数 motion に代入する。ファイルが存在しなかった場合に偽を返す。以前の motion のデータは、ファイルの読み込み前にすべてクリアされる。

bool saveMotionToFile(const char *name);

メンバ変数 motion の内容を name というファイルに保存する。

bool saveMotionToFile(const char *name,const char *dir);

メンバ変数 motion の内容を dir というディレクトリの下に name というファイルに保存する。

bool loadMotionFromMseqFile(const char *name, SerialRobot *r);

Yaml 形式の動作列ファイル name を読み込み、メンバ変数 motion の内容にセットする。このファイルは、Choreonoid で使用されているキーポーズ列と似ているが、各関節角の目標値ではなく、増分をあらわしたものである。

bool loadMotionFromPseqFile(const char *name, SerialRobot *r);

Choreonoid で使用されている Yaml 形式のキーポーズファイル name を読み込み、メンバ変数 motion の内容にセットする。

bool loadMotionFromYamlFile(const char *name, SerialRobot *r);

Choreonoid で使用されている Yaml 形式の動作列ファイル name を読み込み、メンバ変数 motion の内容にセットする。

bool saveMotionToPseqFile(const char *name, SerialRobot *r);

メンバ変数 motion の内容を Choreonoid で使用されているキーポーズ列の Yaml 形式に変換して name というファイルに保存する。（未実装）

bool saveMotionToYamlFile(const char *name, SerialRobot *r);

メンバ変数 motion の内容を Choreonoid で使用されている動作列の Yaml 形式に変換して name というファイルに保存する。（未実装）

void printMotion();

メンバ変数 motion の内容を標準エラー出力に出力する。

void printMotion(std::ofstream& o);

メンバ変数 motion の内容をファイルストリーム o に出力する。この関数は、メンバ関数 saveMotinToFile の内部で呼び出されている。

void clear();

メンバ変数 motion の要素をすべて削除する。

bool appendMotion(RobotMotion * rm);

メンバ変数 motion の要素に、引数の RobotMotion のメンバ変数 motion の内容を追加する。

RobotMotion *dupMotion();

自身のコピーを作成し、それを返す。

class SerialRobot:

【クラスの概要】

このクラスは、シリアルポート経由で動作制御を行うロボットの規定クラスである。個々のロボットを制御するクラスは、このクラスの子クラスとして作成する。このクラスでは、ロボットへのコマンド送信および現在の姿勢を取得は、別スレッド（以下、制御スレッドと呼ぶ）を起動し、処理するようになっている。

【メンバ変数】

Public 型:

char *name;

ロボットの名前

int joints;

ロボットの関節数

char *servoState;

ロボットの関節サーボモータの状態

RobotPosture *initPosture;

ロボットの初期位置の姿勢。デフォルトでは、すべての関節角を 0 とする。

RobotPosture *currentPosture;

ロボットの現在の姿勢。ロボットへ各関節角を問い合わせその結果を保持する。

RobotPosture *targetPosture;

ロボットの目標姿勢。このメンバ変数で目標姿勢の設定を行い、メンバ関数 startMotion でロボットへ目標姿勢とその動作時間を送信する。

std::string motionDir;

ロボットのモーションファイルが格納されているディレクトリ。モーションの読み込み／書き込みは、このディレクトリの下で行われる。

RobotMotion *motion;

現在のロボットの動作。

Private 型:

int threadLoop;

制御スレッドのループに関するフラグ。この値が 1（初期値）である時、制御スレッドは動作しており、終了時には個の値を 0 に変更することで、制御スレッドのループを停止させる。

int motionTime;

このメンバ変数は、デフォルトの目標姿勢への動作時間を表す。単位は、ミリ秒である。この変数を変更したい場合には、メンバ関数 setDefaultMotionTime を使う。

int senseTime;

このメンバ変数は、ロボットへ現在の姿勢を問い合わせる場合のインターバルを表す。単位は、ミリ秒である。本来、高機能の制御装置であれば、制御スレッドの制御時間ごとに現在の姿勢を問い合わせるべきであるが、低速のシリアルインターフェースを使用するロボットでは、毎回すべての関節角の状態の問い合わせには多くの時間を要するため、動作コマンドを送信していない時間に現在の姿勢を問い合わせるように制御スレッドを実装している。

int timeout;

このメンバ変数は、制御ループのインターバルの時間を表す。この値は、連続的な動作の時には、RobotPosture のメンバ変数である motionTime が代入され、通常時は、メンバ変数 senssTime が代入される。

int repeatCount;

このメンバ変数は、ロボットの現在の動作（メンバ変数 motion）の繰り返し再生回数を示す。

int reverseFlag;

このメンバ変数は、ロボットの現在の動作（メンバ変数 motion）の再生時に、逆向きに再生するかどうかを設定する。

bool executeMotion;

このメンバ変数は、ロボットの現在の動作（メンバ変数 motion）の再生中かどうかを表す。

int commandCount;　　// number of the commands　to send

このメンバ変数は、制御ループにおいてロボットの動作指令(targetPosture)を送信する残り回数を表している。すなわちこの変数が、0 以上の時、targetPosture に格納された姿勢へ動くためのコマンドがロボットへ送信される。制御ループでは、1 つのコマンドを送信終了後、この変数を 1 減じる。

char *commandBuf;

この変数は、ロボットへのコマンド送信のためのバッファである。制御スレッド内では、メンバ変数 targetPosture へ移動するコマンドを変換後、この変数にコピーしてロボットへ送信する。デフォルトでは、この変数の長さは最大１２８バイトとしている。

int commandSize;

この変数は、ロボットへの送信するコマンドの長さを表す。すなわち、メンバ変数 commandBuf に格納されたロボットへのコマンドの長さになる。

THREAD_HANDLE hThread;

制御スレッドへのハンドラ。

MUTEX_HANDLE mutex_com;

ロボットへコマンドを送信する場合のシリアルポートを排他制御するためのミューテックス。

MUTEX_HANDLE mutex_motion;

ロボットへの送信するコマンドバッファ(メンバ変数 commandBuf)の操作など、制御スレッド内で参照、変更を排他制御するためのミューテックス。

SerialCom *com;

ロボットへコマンド送信するためのインタフェース。

【コンストラクタとデストラクタ】

コンストラクタ：コンストラクタでは、ロボットを接続しているシリアルポート名と関節角数を引数として与えなければならない。この引数に従って、メンバ変数の初期化を行う。

- SerialRobot(char *devname, int n);

デストラクタ：デストラクタでは、制御スレッドを停止し、その後、シリアルポートの削除、RobotMotion の削除、初期化姿勢、現在の姿勢、目標姿勢などをメンバ変数の削除を行う。

- ~SerialRobot();

【メンバ関数】

Public 型：

int openPort();

ロボットと通信するためのシリアルポートを開く。内部では、com->openPort()を実行し、その結果を返す。

void closePort();

ロボットと通信するためのシリアルポートを閉じる。内部では、com->closePort()を実行する。

void setDevice(char *devname);

ロボットと通信するためのシリアルポート名を devname に設定する。内部では、com->setDevPort(devname)が実行される。

void startMotion();

メンバ変数 commandCount を 1 に設定する。これによって制御スレッドでは、targetPosture の姿勢へ移動するためのコマンドを生成してロボットへ送信することができる。

int initPosition();

ロボットの初期姿勢である initPosture を targetPosture にコピーし、メンバ関数 startMotion を実行する。これによって、ロボットを初期姿勢にすることができる。

int appendCurrentPosture();

現在の姿勢（メンバ変数 currentPosture）を動作 motion の末尾に追加する。動作の時間は、メンバ変数 motionTime とする。

int appendCurrentPosture(double mtime);

現在の姿勢（メンバ変数 currentPosture）を動作 motion の末尾に追加する。動作の時間は、mtime とする。

int loadMotion(char *name);

メンバ変数 motionDir の下でファイル name.m, name.yaml, name.pseq, name.msep の順で検索し、ファイルが存在すれば、そのファイルから動作を読み込み、メンバ変数

motion に入れ、motion の長さを返す。どのファイルも存在しない場合は、-1 を返す。このとき、motion はクリアされている。

int loadMotionFromM(char *fname);

メンバ変数 motionDir の下でファイル fname.ファイルから動作を読み込み、メンバ変数 motion に入れ、motion の長さを返す。このファイルは独自形式であり、読み込みに失敗した場合は -1 を返す。このとき、motion はクリアされている。

int loadMotionFromYaml(char *fname);

メンバ変数 motionDir の下でファイル fname.ファイルから動作を読み込み、メンバ変数 motion に入れ、motion の長さを返す。このファイルは Choreonoid の動作列をあわす Yaml 形式のファイルであり、読み込みに失敗した場合は -1 を返す。このとき、motion はクリアされている。

int loadMotionFromPseq(char *fname);

メンバ変数 motionDir の下でファイル fname.ファイルから動作を読み込み、メンバ変数 motion に入れ、motion の長さを返す。このファイルは Choreonoid のキーポーズ列をあわす Yaml 形式のファイルであり、読み込みに失敗した場合は -1 を返す。このとき、motion はクリアされている。

int loadMotionFromMseq(char *fname);

メンバ変数 motionDir の下でファイル fname.ファイルから動作を読み込み、メンバ変数 motion に入れ、motion の長さを返す。このファイルは Choreonoid のキーポーズ列をあわす Yaml 形式のファイルと似ているが、関節角の目標値ではなく関節角の増分がきさいされている。読み込みに失敗した場合は -1 を返す。このとき、motion はクリアされている。

int saveMotionToM(char *fname);

現在登録されている動作（メンバ変数 motion）をファイルに保存する。

int saveMotionToYaml(char *fname);

現在登録されている動作（メンバ変数 motion）を Choreonoid で使われている動作列を表す Yaml 形式ファイルに変換し、保存する。（未実装）

int saveMotionToPseq(char *fname);

現在登録されている動作（メンバ変数 motion）を Choreonoid で使われているキーポーズ列を表す Yaml 形式ファイルに変換し、保存する。（未実装）

int setJoint(unsigned char id, short deg);

目標姿勢のメンバ変数 targetPosture に対して id に対応する関節角に deg 度の値をせっていする。id は、1 から numJoints までの整数としている。

int setMotionTime(int tm);

メンバ変数 targetPosture の移動動作時間を tm にする。tm は、msec の単位で指定されている。このデフォルト動作時間は、RobotPosture の motionTime=0 のときに使用される。

void clearMotion();

メンバ変数 motion の要素をすべて削除する。

int setDefaultMotionTime(int sval);

メンバ変数 motionTime に sval を設定し、その値を返す。sval は、msec の単位で指定されており、このデフォルト動作時間は、RobotPosture の motionTime=0 のときに使用される

int getDefaultMotionTime();

メンバ変数 motionTime の値を返す。

int setTimeout(intval);

メンバ変数 timeout に val を設定し、その値を返す。

int getTimeout();

メンバ変数 timeout の値を返す。

int setCommand(char *packet, intlen);

メンバ変数 commandBuf に、ロボットへのコマンド packet をコピーし、コマンドの長さ len を commandSize に代入する。

int recieveData(char *data, intlen);

シリアルポート（ロボット）からのデータを受信し、data に代入する。データのバイト数は、len に代入する。

int sendCommand(char *data, intlen);

シリアルポート（ロボット）へコマンド data を送信する。

int startThread();

制御スレッドを開始する。

int stopThread();

制御スレッドを停止させる。すなわち、メンバ変数 threadLoop に 0 を設定し、制御スレッドを終了させる。

int isActive();

制御スレッドの状態（メンバ変数 threadLoop）の値を返す。

RobotPosture *getNthPosture(int n);

メンバ変数 motion に格納された姿勢列のうち、メンバ変数 reverseFlag が真の時には、最後尾から n 番目の姿勢を、偽の時には前から n 番目の姿勢を返す。n が motion の長さより大きい場合には、NULL を返す。

int doNthMotion(int n);

メンバ変数 motion に格納された姿勢列のうち、メンバ変数 reverseFlag が真の時には、最後尾から n 番目の姿勢を、偽の時には前から n 番目の姿勢を目標姿勢 tergetPosture にコピーし、その姿勢に移動する。このとき移動時間は、メンバ変数 motionTime とする。

RobotPosture *getTargetPosture();

目標姿勢を表すメンバ変数 tergetPosture を返す。

RobotPosture *getCurrentPosture();

現在の姿勢を表すメンバ変数 currentPosture を返す。

RobotPosture *getFirstPosture();

メンバ変数 motion に格納された姿勢列のうち、メンバ変数 reverseFlag が真の時には、最後尾の姿勢を、偽の時には、最初の姿勢を返す。

RobotPosture *getLastPosture();

メンバ変数 motion に格納された姿勢列のうち、メンバ変数 reverseFlag が真の時には、最初の姿勢を、偽の時には、最後尾の姿勢を返す。

int isMoving();

メンバ変数 commandCount が正の時には 1 を、それ以外では 0 を返す。

int setMotionCount(int count);

メンバ変数 repeatCount に count を設定し、その値を返す。

int numJoints();

メンバ変数 joints の値を返す。

char *getServoState();

メンバ変数 servoState を返す。

int setServoState(int id, int state);

メンバ変数 servoState の　id-1 番目の要素を state にし、その値を返す。

virtual　int checkConnection();

設定しているシリアルポートの確認を行う。接続が成功していれば、0 を返すようにする。この関数は、子クラスで上書きすべきである。このクラスでは、常に 0 を返すように実装されている。

virtual short getAngle(unsigned char id);

id に対応する関節角を返す。関節角の取得に失敗すれば、-1000 を返す。このクラスでは、ロボットの実装がないため常に-1000 を返す。子クラスを作るときには、必ず override すること。

int connect();

ロボットへのシリアルポートを開いて、接続を確認する。

virtual void getPosture();

ロボットから現在の姿勢を読み込み、メンバ変数 currentPosture に代入する。

virtual void svc();

制御スレッド内の制御ループの中で呼び出す。この関数が制御スレッドのメイン関数である。OpenRTM-aist における onExecute に該当する。

virtual void postureToCommand(RobotPosture *pos) = 0;

引数で与えられた RobotPosture をロボットへの動作制御コマンドに変換する。変換された制御コマンドは commandBuf、コマンド長は commandSize に代入する。

virtual int jointIdToMotorId(int jid) = 0;

引数で与えられたジョイント ID をモータ ID に変換する。この関数は、対象となるロボットに依存する。

virtual int motorIdToJointId(int mid) = 0;

引数で与えられたモータ ID をジョイント ID に変換する。この関数は、対象となるロボットに依存する。

virtual int stabilizer() = 0;

目標関節角へ移動する場合、targetPosture に対するキャリブレーションを行うための関数である。この関数は必須ではないが、安定歩行動作等を実施する場合に各自実装すること。デフォルトでは何もしない関数が実装すべきである。

void setMotionDir(const char *dir);

動作を読み込む場合のディレクトリを指定する。すなわち、引数 dir はメンバ変数 motionDir にコピーされる。

GROBO.h

class GR001：

【クラスの概要】

このクラスは、HPI 社製人間型ホビーロボット GR001 を制御するためのものであり、前述の SerialRobot クラスの子クラスとして実装されている。

【メンバ変数】

Private 型：

RobotMotion *record;

記憶用の動作列。

short *Volts;

各モータの電圧を保持するための配列。

short *Currents;

各モータの電圧を保持するための配列。

unsigned char *MoveJointPacket;

１つのモータに対する動作命令のためのショートパケットのテンプレート。

unsigned char *PosturePacket;

すべてのモータの動作命令のためのロングパケットのテンプレート。

unsigned char *MotorInfoPacket;

１つのモータの状態を取得するためのリクエストパケットのテンプレート。

unsigned char *ServoPacket;

１つのモータのサーボ状態を設定するためのショートパケットのテンプレート。

std::map<std::string, std::vector<int>>m_jointGroups;

関節 ID のグループかのためのテープル（動作未確認）

std::map<std::string, int> Link;

関節 ID に名前付けするためのテーブル（動作未確認）

【コンストラクタとデストラクタ】

コンストラクタ：コンストラクタは、接続するシリアルデバイス名を与え、メンバ変数 Volts, Currents, MoveJointPacket, PosturePacket, MotorInfoPacket, ServoPacket の領域確保と初期化を行う。

- GR001(char *devname);

デストラクタ：デストラクタでは、制御スレッドを停止し、その後、シリアルポートの削除、RobotMotion の削除、初期化姿勢、現在の姿勢、目標姿勢などをメンバ変数の削除を行う。

- ~GR001();

【メンバ関数】

Public 型：

int connect(int n);

ロボット制御用のシリアルポートへの接続を行う。n=0 の場合には、COM0（UNIX 系 OS では、/dev/ttyS0）から順にデバイスファイルをオープンし、メンバ関数 checkConnection を用いて接続が確認できれば、デバイス名と通信ハンドラ等を設定し、0 を返す。失敗した場合には、−1 を返す。

int connect();

ロボット制御用のシリアルポートへの接続を行う。これは、親クラスである SerialRobot のメンバ関数 connect()を呼び出しているのみである。

int checkConnection();

オープンしたシリアルポートに対して{0x41, 0x00}のショートパケットを送信し、0x07 が返ってきた場合には、GR001 のモータコントローラであると判断し、0 を返す。それ以外では、-1 を返す。

int printVolts(int flag);

メンバ変数 Volts から、平均のモータ電圧を計算しその値を返す。引数 flag が 0 以外の時には、標準エラー出力に各モータの電圧を出力する。メンバ変数 Volts の更新は、メンバ関

数 getPosture を実行した時に、同時に取得される。

void printCurrents(int flag);

メンバ変数 Currents を標準エラー出力に出力する。メンバ変数 Currents の更新は、メンバ関数 getPosture を実行した時に、同時に取得される。

void printPosture();

ロボットの現在の姿勢を表すメンバ変数 currentPosture の要素を標準エラー出力に主力する。

int setTorque(unsigned char id, char val);

id で指定されたモータのサーボの状態を変更する。val=1 の場合にはサーボをオンにし、0 の場合には、サーボをオフにする。なお、id は、モータ ID であり、1 から numJoints までの整数である。

void setServo(char on, int c);

モータのサーボ状態を変更する。on=1 の場合には、サーボオンにし、0 の場合には、サーボオフにする。引数 c は、モータ ID でなく、下記のモータのグループになる。

c=0→　すべてのモータ
c=1　→頭部と胴体のモータ
c=2　→右腕のモータ
c=3　→左腕のモータ
c=4　→両腕のモータ
c=5　→右足のモータ
c=6　→左足のモータ
c=7　→両足のモータ

short getAngle(unsigned char id);

モータ ID ＝id のモータの現在の角度を問い合わせその値を返す。問い合わせに失敗した場合は、-254 を返す。

int setDegMs(char *data, short deg, unsigned short cs);

モータ制御用コマンド data に対し、目標角度とその動作時間を設定する。目標角度は、セ

ンチ度、動作時間は、センチ秒を単位とする。例えば、90 度を 100 ミリ秒で動作させたい場合には、deg =900, cs = 1000 となる。

int moveJoint(unsigned char id, short deg, unsigned short cs);

モータ ID=id のモータを deg の角度まで cs 時間で動作させる。このコマンドは、ショートパケットを用いて即座にロボットへ送信される。deg, cs に関しては、メンバ関数 setDegMs の引数と同じである。

int setJoints(short *deg, intcs);

引数の deg で表された目標角度と動作時間を targetPacket に設定する。引数の配列 deg は、ロボットの関節角数以上の長さを持っていなければならない。（エラー処理が省略されている）

int setJointsFromString(char *str);

引数 str で指定された目標角度および動作時間を targetPacket に設定する。Str は、スペースで区切られた整数の列であることを仮定している。また、最初の数は、motionTime であり、すべての関節角を指定するには、関節角数+1 の整数が必要である。

int activateMotionSlot(unsigned char id);

GR001 のコントローラにあらかじめ記憶されている id の動作を実行する。（動作未確認）

int activateSenario(unsigned char id);

GR001 のコントローラにあらかじめ記憶されている id のシナリオを実行する。（動作未確認）

void postureToCommand(RobotPosture *pos);

引数の RobotPosturepos を目標姿勢とするように、GR001 の動作コマンドに変換し commandBuf にコマンドの長さを commandSize に格納する。GR001 の制御 CPU のファームウェアが変更された場合には、この関数を修正するのみで動作するはずである。

double *getCurrentJointAngles();

現在の姿勢を取得し、その関節角の列（単位は radian）を返す。ただし、返さえる配列は、モータ ID の順に並べてある。

int setTargetJointAngles(double *rad);

ラジアン表現された関節角の列をメンバ変数 tragetPosture に設定し、実行する。ただし、与えられる目標関節角の配列は、モータ ID の順に並べてあることを前提にしている。

int jointIdToMotorId(int jid);

引数で与えられたジョイント ID をモータ ID に変換し、その値を返す。

int motorIdToJointId(int mid);

引数で与えられたモータ ID をジョイント ID に変換し、その値を返す。

int stabilider();

目標関節角の補正用関数であるが、現在は何もしない。

int clearRecord();

メンバ変数 record の動作列をクリアする。

int recordCurrentPosture();

現在の姿勢をメンバ変数 record の動作列に追加する。

int doRecord();

メンバ変数 record の動作を再生する。

以下のメンバ関数は、hrpsys-base のロボットクラスと互換性を実現するためのものである。(動作未確認)

bool servo(intjid, boolturnon);

jid のモータの状態を変更する。

bool servo(const char *jname, boolturnon);

jname で指定されたモータの状態を変更する。

void readJointAngles(double *o_angles);

モータの現在の関節角(メンバ変数 currentPosture)を o_angle にコピーする。角度の単はラジアンとする。

void writeJointCommands(const double *i_commands);

i_commads で指定された目標関節角をメンバ変数 targetPosture に設定する。

bool checkEmergency(std::string &o_reason, int&o_id);

モータのエラー状態を検出する。未実装。

void oneStep();

メンバ変数 targetPosture を目標姿勢として、動作させる。メンバ関数 startMotor と同じ。

bool addJointGroup(const char *gname, conststd::vector<std::string>&jnames);

関節のグループ化と名前付けを行う。

Private 型：

bool names2ids(conststd::vector<std::string>&i_names, std::vector<int>&o_ids);

i_names で与えられた関節名をモータ ID に変換する。