管理 No. AE21-022-00

ロボットハンド
ROS パッケージ

取扱説明書

目次

１　はじめに

■本パッケージについて

PLEXMOTION ブランドにて販売しておりますロボットハンドを ROS 上で制御するノードです。
本パッケージを使用する際は、必ず本取扱説明書の記載条件をご確認ください。本パッケージをご使用された場合は、本取扱説明書の記載条件に同意されたものとみなします。

■対象製品

電動 3 爪ロボットハンド　ARH305A
電動 3 爪ロボットハンド　ARH305B
電動 3 爪ロボットハンド　ARH350A

■動作確認済環境

・ＯＳ
Ubuntu16.04, Ubuntu18.04, Ubuntu20.04, Raspbian(Raspberry Pi4)

・ＲＯＳディストリビューション
Kinetic, Melodic, Noetic

■対象製品の使用方法について

対象製品の取扱説明書をご参照ください。

■ロボットハンド制御パッケージ

aspina_hand.zip　：　ZIP ファイルにてご提供いたします。

２　ご注意事項

２．１　免責事項

（１）　お客様がご自身で選択される本パッケージを含むその他システムに適用される規格、それらに関する法令または規制、および下記（４）（５）のソフトウェアに関する条件に関しましては、お客様ご自身でのご確認および遵守をお願いいたします。

（２）　（１）に基づき、お客様の本ソフトウェアの使用に起因して第三者の権利の侵害および損害が発生した場合は、シナノケンシ株式会社は一切の責任を負いません。

（３）　本パッケージは、シナノケンシ株式会社によって現状有姿のまま提供され、動作不良、不具合、第三者権利侵害、商品性および特定の目的に対する適合性または有用性に関する明示または黙示の保証は一切含まれません。いかなる場合も、本パッケージを使用した結果または使用できなかったことによることについての直接的、間接的、偶発的、例示的、または結果的損害（代替商品またはサービスの購入、使用、データ、または利益の損失を含むがこれらに限定されない）に対して、そのような損害の可能性を知らされていた如何にかかわらずシナノケンシ株式会社は責任を負わないものとします。

（４）　本パッケージは、BSD License に基づきライセンスされるソフトウェアが含まれています。当該ソフトウェアに関する条件は、お客様ご自身でご確認ください。
https://opensource.org/licenses/bsd-license.php

（５）　本パッケージは、GNU Lesser General Public License（LGPL）に基づきライセンスされるソフトウェアを使用しています。当該ソフトウェアに関する条件は、お客様ご自身でご確認ください。
https://www.gnu.org/licenses/lgpl-3.0.en.html

（６）　本説明書を、シナノケンシ株式会社の許可なしに複写、複製、再配布する事を禁じます。

（７）　本パッケージの使用に対する許諾は、シナノケンシ株式会社の著作権およびその他の知的財産権の譲渡または実施権の許諾をともなうものではありません

２．２　使用上の注意

（１）　ご使用のロボットハンドによって、開閉時間 runtime の設定範囲が異なります。下限値よりも低い値を設定するとハンドは動作しません。

ARH305A	250（0.25 秒）～ 10000（10 秒）
ARH305B	250（0.25 秒）～ 10000（10 秒）
ARH350A	800（0.80 秒）～ 10000（10 秒）

３　準備

３．１　ハードウェアの準備

①ロボットハンドの取扱説明書に記載されている仕様の電源を準備します。

②ロボットハンドとの通信を行う USB↔RS485 コンバーターなどを準備します。
※ロボットハンドの通信規格は RS485 です。

③ ①②を接続してロボットハンドに電源を投入します。

④ロボットハンドとの通信を行うデバイスを確認します。
システムの /dev のデバイス一覧から、ロボットハンドと接続されている通信デバイスの
デバイス名を確認します。　例）/dev/ttyUSB0 など。

※ロボットハンドには個別の通信 ID があります。
通信 ID は１～に設定してください。※複数のハンドを接続する場合は ID1, ID2 と連番にします。

※通信デバイスがユーザー権限からアクセスできない事があります。
その際は下記の手順で通信デバイスの権限を変更してください。※通信デバイス/dev/ttyUSB0 の例。

```
$ sudo chmod 666 /dev/ttyUSB0
```

３．２　ロボットハンド ROS パッケージのインストール方法

ご使用 OS に対応した ROS ディストリビューションはインストール済みとし、新規に
ROS ワークスペースを作成する手順からご説明いたします。

① ユーザー・ワークスペースの作成 ※説明ではワークスペース名を catkin_ws としています。

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ..
$ catkin_make
```

上記は以下の作業を行っています。
・ワークスペースのフォルダとパッケージを追加する src フォルダを作成
・src フォルダに移動
・ワークスペース初期化
・ワークスペースのルートフォルダに移動
・ワークスペース作成（ビルド）

② ロボットハンド制御パッケージの追加とビルド
ロボットハンド制御パッケージが格納されたファイルを /home/aspina_hand.zip とします。

```
$ unzip /home/aspina_hand.zip -d ~/catkin_ws/src
$ cd ~/catkin_ws
$ source devel/setup.bash
$ catkin_make
```

上記は以下の作業を行っています。
・ロボットハンド制御パッケージを展開
・ワークスペースのルートフォルダに移動
・ワークスペースの環境設定
・ビルド

以上でパッケージ（制御ノード、テストノード）のインストールが完了します。

３．３　動作確認

ロボットハンド制御パッケージにあるテストノードを使って動作確認します。
テストノードはハンドの爪を繰り返し全開・全閉します。

① ROS コアの実行
端末（ターミナル）を起動し、以下の操作を行います。

```
$ roscore &
```

上記は以下の作業を行っています。
・ROS コアをバックグラウンドで実行

② ロボットハンド制御ノードの起動
新しい端末（ターミナル）を起動し、以下の操作を行います。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ rosrun aspina_hand aspina_hand_node
```

上記は以下の作業を行っています。
・ワークスペースに移動
・ワークスペースの環境設定
・パッケージ aspina_hand のロボットハンド制御ノード起動

ロボットハンド制御ノードの標準通信デバイスは /dev/ttyUSB0 となっています。
通信デバイスが異なる場合は、rosrun の行を下記の様にします。
例）通信デバイスが /dev/ttyS0 の場合。 -d オプションを使います。

```
$ rosrun aspina_hand aspina_hand_node -d /dev/ttyS0
```

③ ロボットハンド動作テストノードの起動。
新しい端末（ターミナル）を起動し、以下の操作を行います。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ rosrun aspina_hand aspina_hand_test
```

上記は以下の作業を行っています。
・ワークスペースに移動
・ワークスペースの環境設定
・パッケージ aspina_hand のロボットハンドテストノード起動

テストノードによってロボットハンドが開閉動作を繰り返します。
停止させる場合は Ctrl キーを押しながら C キーを押します。

３．４　ノードの起動オプション

ロボットハンド制御ノードとテストノードには幾つかの起動オプションがあります。

ROS 上でのノード起動は rosrun コマンドを使います。
rosrun コマンドのパラメータは以下です。

rosrun パッケージ名 ノード名 [ノードオプション 1] [ノードオプション 2]・・・

※ノードのオプションはノード毎に異なります。

① 制御ノードのオプション
制御ノードでは通信デバイスと管理するハンド数をオプションで指定できます。
指定しない場合の標準は、通信デバイス /dev/ttyUSB0 、ハンド数 １ となります。

・通信デバイス指定オプション –d
以下の様に指定します。

rosrun aspina_hand aspina_hand_node –d デバイス名

・管理するハンド数指定オプション –h
以下の様に指定します。管理可能なハンド数は 1～最大 31 です。

rosrun aspina_hand aspina_hand_node –h ハンド数

※複数のハンドを管理する場合は、それぞれのハンドの通信 ID が１～順になる様に設定し、
末端のハンドのみ終端抵抗をオンにした上で、ハンドを通信回線に並列接続してください。

※制御ノードは複数のハンドを管理できますが、受付可能なサービスとメッセージはそれぞれ
１つです。制御ノードへのサービスコール、メッセージ発行するノードは一つにして頂くか、
同時にサービスコールやメッセージ発行が行われない様に排他制御をお願いいたします。

② テストノードのオプション
テストノードでは、制御ノードへの要求方法（サービスによる制御、メッセージによる制御）と、テストするハンド数をオプションで指定できます。
指定しない場合の標準は、要求方法がサービス、ハンド数が１となります。

・テストするハンド数指定オプション –h
以下の様に指定します。テスト可能なハンド数は 1～最大 31 です。
制御ノードで指定したハンド数と同じにします。

```
rosrun aspina_hand aspina_hand_test –h ハンド数
```

・要求方法オプション –t
-t オプションが指定された場合、制御ノードへの要求はトピック（メッセージ）で行われます。

```
rosrun aspina_hand aspina_hand_test –t
```

※サービスによる要求に対してトピックによる要求は ROS に起因する遅延が発生し、
どの程度の遅延が生じるかは動作環境によって異なります。
サービス、トピックそれぞれの特徴については、第４章「サービス、メッセージによる
ロボットハンドの制御」をご参照ください。

４　サービス、メッセージによるロボットハンドの制御

ロボットハンド制御ノードへサービスコールまたはメッセージ（トピック）を発行する事で
ロボットハンドを制御します。
本章では C++言語によるプログラミングでロボットハンドを制御する方法を解説いたします。

※サービスコール、メッセージ発行を行う前に制御ノードを起動してください。

４．１　サービスとメッセージの違いと特徴

２つの制御方法には一長一短がございますので、お客様の環境に応じて適宜ご選択ください。

・サービスによる制御
プログラミング言語の関数コールに近い制御方法です。
サービスによる制御要求は直ちに制御ノードに伝えられ、制御ノードで処理が行われた後に
結果が返ります。
制御要求に対する遅延が少なく制御も容易ですが、制御ノードが処理を実行して結果を
返すまでサービスコールが終了しないブロッキング動作となりますので、一つのループで複数の
装置（モータなど）を同時に制御する様なノードには不向きです。

・メッセージ（トピック）による制御
一般的な通信による制御に近い方法です。
制御メッセージの発行は制御ノードが講読したかに関わらず直ちに終了します。
結果は制御ノードが制御メッセージに対応した処理を行い、結果メッセージを発行する事で
伝えられます。
制御ノードの動作に関わらずノンブロッキングでの制御が可能なので、一つのループで複数の装置を
同時に制御する様なノードに向いています。

４．２　サービスとメッセージの定義ファイル

ロボットハンドを制御するサービスとメッセージは「３．２　ロボットハンド ROS パッケージの
インストール方法」でパッケージをインストールする事で、ワークスペースの
共有フォルダ「ワークスペース/devel/include」に「aspina_hand」フォルダが作成され
以下のファイルが登録されます。

① サービス制御用のファイル

・Hand.h

② メッセージ制御用のファイル

・AspinaHandCommand.h
コマンドメッセージ

・AspinaHandStatus.h
ステータスメッセージ

４．３　サービスとメッセージによるロボットハンドの制御

① サービス

一組のリクエスト、レスポンスで制御します。

・サービス名

aspina_hand/hand

・インクルードファイル

aspina_hand/Hand.h

・リクエスト

```
uint8 id
uint8 cmd
uint32 position
uint32 effort
uint32 runtime
```

id ： 制御するハンドの ID(1～31)を指定します。
cmd ： 以下のコマンドコードを指定します。
　　0 を指定した場合は position , effort , runtime に基づいてハンドを動かします。
　　1 を指定した場合は ハンドの状態を返します（position , effort , runtime は使いません)。
position ： ハンドの爪の位置（開閉量）を 0(全開)～1000(全閉)で指定します。
effort ： ハンドの力を 5(最小)～1000(最大)で指定します。
runtime ： フルストロークの動作時間（ミリ秒）を ~~250（0.25 秒）~~～10000(10 秒)で指定します。

・レスポンス

```
uint8 stalled
uint8 reached_goal
uint32 position
uint32 effort
```

stalled ： ハンドに異常がなければ 0、ハンドに異常がある場合は 0 以外になります。
reached_goal ： ハンドが動作中なら 0、待機中なら 0 以外になります。
position ：ハンドの爪の現在位置（0(全開)～1000(全閉)）です。
effort ：ハンドが出している現在の力（0(最小)～1000(最大)）です。

・C++言語でサービスを使って制御する例
ID１のロボットハンドをサービスで開閉させます。

例１ sample_4-3-1.cpp

```
#include <ros/ros.h>
#include <sstream>
#include <unistd.h>
#include <time.h>
#include  “aspina_hand/Hand.h”

static ros::ServiceClient mClient;

static bool WaitForIdling(void);
static bool Action(bool close);

int main(int argc, char **argv)
{
  // ROS 初期化
  ros::init(argc, argv, "test");
  ros::NodeHandle n;

  // サービスクライアント作成
  mClient = n.serviceClient<aspina_hand::Hand>("aspina_hand/hand");

  // 開閉テスト ※前提としてプログラム実行時のハンドの爪は全開している事。
  while(ros::ok())
  {
    // 待機待ち
    if(WaitForIdling() == false)
    {
      break;
    }
    // 閉
    if(Action(true) == false)
    {
      break;
    }
    // 待機待ち
    if(WaitForIdling() == false)
    {
      break;
    }
    // 開
    if(Action(false) == false)
    {
      break;
    }
  }

  return 0;
}
```

```
// ハンド待機待ち
static bool WaitForIdling(void)
{
  aspina_hand::Hand srv;
  struct timespec time;
  time.tv_sec = 0;
  time.tv_nsec = 10e-3 / 1e-9;
  bool ret = false;
  for(;;)
  {
    srv.request.id = 1;
    srv.request.cmd = 1;
    if(mClient.call(srv) != 0)
    {
      if(srv.response.stalled != 0)
      {
        break;
      }
      else
      {
        if(srv.response.reached_goal != 0)
        {
           ret = true;
           break;
        }
      }
    }
    else
    {
      break;
    }
    nanosleep(&time,NULL);
  }
  return ret;
}

// アクション実行
static bool Action(bool close)
{
  aspina_hand::Hand srv;
  srv.request.id = 1;
  srv.request.cmd = 0;
  srv.request.position = (uint32_t)(close == true ? 1000 : 0);
  srv.request.effort = 1000;
  srv.request.runtime = 250;
  if(mClient.call(srv) != 0)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

②メッセージ（トピック）
ロボットハンドを制御するコマンドメッセージと状態を返信するステータスメッセージがあります。

・コマンドメッセージ（お客様のノード(Publish)　→　ロボットハンド制御ノード(Subscribe)）
aspina_hand/cmd

・インクルードファイル
aspina_hand/AspinaHandCommand.h

・コマンドメッセージの内容
uint8 id
uint8 cmd
uint32 position
uint32 effort
uint32 runtime

id ： 制御するハンドの ID(1～31)を指定します。
cmd ： 以下のコマンドコードを指定します。
0 を指定した場合は position , effort , runtime に基づいてハンドを動かします。
1 を指定した場合は ハンドの状態をステータスメッセージで返します。
position ： ハンドの爪の位置（開閉量）を 0(全開)～1000(全閉)で指定します。
effort ： ハンドの力を 5(最小)～1000(最大)で指定します。
runtime ： フルストロークの動作時間（ミリ秒）を ~~250（0.25 秒）~~～10000(10 秒)で指定します。

・ステータスメッセージ（ロボットハンド制御ノード(Publish)　→　お客様のノード (Subscribe)）
aspina_hand/status

・インクルードファイル
aspina_hand/AspinaHandStatus.h

・ステータスメッセージの内容

uint8 id
uint8 result
uint8 stalled
uint8 reached_goal
uint32 position
uint32 effort

id ： ハンドの ID です。
result ： コマンドの実行が成功した場合は 0、失敗した場合は 0 以外になります。
stalled ： ハンドに異常がなければ 0、ハンドに異常がある場合は 0 以外になります。
reached_goal ： ハンドが動作中なら 0、待機中なら 0 以外になります。
position ：ハンドの爪の現在位置（0(全開)～1000(全閉)）です。
effort ：ハンドが出している現在の力（0(最小)～1000(最大)）です。

・C++言語でメッセージを使って制御する例
ID1 のロボットハンドをメッセージで開閉させます。

※環境に依存する可能性はありますが、ROS 初期化後の最初のメッセージが配信されない症状が発生する事があります。この様な症状が発生した場合はロボットハンド制御ノードからステータスが発行されるまで、ステータス送信コマンドを繰り返し発行してください。

例 2 sample-4-3-2.cpp

```
#include <ros/ros.h>
#include <sstream>
#include <unistd.h>
#include <time.h>
#include  “aspina_hand/AspinaHandCommand.h”
#include  “aspina_hand/AspinaHandStatus.h”

static bool WaitForIdling(void);
static bool Action(bool close);

typedef struct
{
  bool available;
  uint8_t id;
  uint8_t result;
  uint8_t stalled;
  uint8_t reached_goal;
  uint32_t position;
  uint32_t effort;
} HAND_STATUS;

static ros::Publisher mPub;
static ros::Subscriber mSub;
static HAND_STATUS mStatus;

static void Callback(const ros::MessageEvent<aspina_hand::AspinaHandStatus const> &event)
{
  int id = event.getMessage()->id;
  if(id == 1)
  {
    HAND_STATUS *s = &mStatus;
    s-> position = event.getMessage()->position;
    s-> effort = event.getMessage()->effort;
    s-> result = event.getMessage()->result;
    s-> reached_goal = event.getMessage()->reached_goal;
    s-> stalled = event.getMessage()->stalled;
    s->available = true;
  }
  else
  {
    // do nothing
  }
}
```

```
int main(int argc, char **argv)
{
  // ROS 初期化
  ros::init(argc, argv, "test");
  ros::NodeHandle n;
  ros::Rate loop_rate(1000);

  // パブリッシャー、サブスクライバー作成
  mPub = n.advertise<aspina_hand::AspinaHandCommand>("aspina_hand/cmd", 1);
  mSub = n.subscribe("aspina_hand/status",1, Callback);

  // 制御ノードが応答するまで待つ。リトライ回数５。
  int i;
  for(i = 0;i < 5;i++)
  {
    if(WaitForIdling() == true)
    {
      break;
    }
  }
  if(i >= 5)
  {
    return -1;
  }

  // 開閉テスト ※前提としてプログラム実行時のハンドの爪は全開している事。
  while(ros::ok())
  {
    // 待機待ち
    if(WaitForIdling() == false)
    {
      break;
    }
    // 閉
    if(Action(true) == false)
    {
      break;
    }
    // 待機待ち
    if(WaitForIdling() == false)
    {
      break;
    }
    // 開
    if(Action(false) == false)
    {
      break;
    }
  }

  return 0;
}
```

```
// ハンド待機待ち
static bool WaitForIdling(void)
{
  struct timespec time;
  time.tv_sec = 0;
  time.tv_nsec = 10e-3 / 1e-9;
  aspina_hand::AspinaHandCommand cmd;
  cmd.id = 1;
  cmd.cmd = 1;
  bool ret = false;
  int i;
  for(;;)
  {
    mStatus.available = false;
    mPub.publish(cmd);
    for(i = 0;i < 100;i++)
    {
      ros::spinOnce();
      if(mStatus.available == true)
      {
        break;
      }
      nanosleep(&time,NULL);
    }
    if(i >= 100 || mStatus.result != 0)
    {
      break;
    }
    if(mStatus.stalled != 0)
    {
      break;
    }
    else
    {
      if(mStatus.reached_goal != 0)
      {
         ret = true;
         break;
      }
    }
  }
  return ret;
}
```

```
// アクション実行
static bool Action(bool close)
{
  mStatus.available = false;

  aspina_hand::AspinaHandCommand cmd;
  cmd.id = 1;
  cmd.cmd = 0;
  if(close == true)
  {
    cmd.position = 1000;
  }
  else
  {
    cmd.position = 0;
  }
  cmd.effort = 1000;
  cmd.runtime = 250;
  mPub.publish(cmd);

  struct timespec time;
  time.tv_sec = 0;
  time.tv_nsec = 10e-3 / 1e-9;
  int i;
  for(i = 0;i < 100;i++)
  {
    ros::spinOnce();
    if(mStatus.available == true)
    {
      break;
    }
    nanosleep(&time,NULL);
  }
  if(i < 100)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

４．４　ロボットハンド制御クラスライブラリ

サービスとメッセージを意識する事なく容易にロボットハンドを制御可能な C++言語の
クラスライブラリをご用意しております。

① クラスライブラリ

ロボットハンド制御ノードのインストールフォルダに格納されておりますので
必要に応じてお客様の環境にファイルをコピーしてお使いください。
「３．２　ロボットハンド ROS パッケージのインストール方法」の通りに制御ノードを
インストールされた場合は下記のフォルダにライブラリが格納されております。

~/catkin_ws/src/aspina_hand/src/aspinaLib/AspinaHand.h

AspinaHand.h をお客様の環境にコピーしてお使いください。

② クラスライブラリの使用方法

サービスによる制御とメッセージによる制御それぞれのクラスがありますので、用途に応じて
使い分けます。

・サービス制御クラス

・クラス名

AspinaHandService

・関数仕様

・コンストラクタ

AspinaHandService(ros::NodeHandle *node)
ROS ノードハンドルのアドレスをコンストラクタに渡してください。

・ハンドの状態を取得

bool GetStatus(int id,AspinaHandService::STATUS *status);
指定された id のハンドの状態を取得して、status に格納します。
取得に失敗した場合は false が返ります。

id　ハンドの ID(1～31)

ステータス構造体

```
typedef struct
{
  bool     stalled;         //!< true : ハンド異常
  bool     reached_goal;    //!< true : 動作完了（待機中）
  uint32_t position;        //!< 現在位置[0.1%]
  uint32_t effort;          //!< 出力（力）[0.1%]
} STATUS;
```

・ハンド・アクション実行

bool Action(int id,uint32_t position,uint32_t effort,uint32_t runTime,
bool wait = false);

指定された id のハンドに対して effort で指定された力（最大）により position の位置へ runTime で指定された時間で開閉します。
アクションの実行に失敗した場合は false が返ります。

id　ハンドの ID(1～31)
position　開閉位置(0:全開～1000:全閉)
effort　出力 (5:最小～1000:最大)
runTime　開閉時間 (~~250:最短~~～10000:最長)　※ミリ秒単位です。
wait　true : 開閉を完了するまで Action 関数は終了しません。

例 3 sample_4-4-2-1.cpp
・サービスによる制御 ※ロボットハンドの開閉を繰り返すサンプルコードです。

```
#include “AspinaHand.h”
int main(int argc, char **argv)
{
  ros::init(argc, argv, "test");
  ros::NodeHandle n;
  AspinaHandService service(&n);
  int id = 1; // Hand ID = 1
  AspinaHandService::STATUS sts;
  while(ros::ok())
  {
    if(service.GetStatus(id,&sts) == true)
    {
      if(sts.stalled == false && sts.reached_goal == true)
      {
        uint32_t pos;
        if(sts.position < 500)
        {
          pos = 1000;
        }
        else
        {
          pos = 0;
        }
        if(service.Action(id,pos,1000,250,false) == false)
        {
          return -1;
        }
      }
    }
    else
    {
      return -1;
    }
  }
  return 0;
}
```

・メッセージ(トピック)制御クラス

・クラス名
AspinaHandTopic

・関数仕様

・コンストラクタ
AspinaHandTopic(ros::NodeHandle *node,int handNumber)
ROS ノードハンドルのアドレスと管理するハンドの数をコンストラクタに渡してください。

handNumber 管理するハンドの数。例）1 を指定した場合は ID1 のハンドのみ管理。
２を指定した場合は ID1 と ID2 のハンドを管理。

・制御開始　※ハンドの状態取得や制御を行う前に各ハンドに対して一度実行して下さい。
bool Startup(int id);
指定された ID のハンドの制御を開始します。
開始に失敗した場合は false が返ります。

id　ハンドの ID(1～31)

・ハンドの状態を取得
int GetStatus(int id,AspinaHandTopic::STATUS *status,bool wait = false);
指定された ID のハンドの状態を取得し、status に格納します。
wait の指定と関数の戻り値の関係は以下になります。(wait=false はノンブロッキングです)。

戻り値	意味
0	状態が status に格納されました。
1	wait=false とした場合、状態が制御ノードから届いていない事を示します。 戻り値が 0 になるまで繰り返し GetStatus を実行して下さい。
-1	エラーが発生しました。

※STATUS 構造体の説明はサービス制御クラスの GetStatus をご参照下さい。

・ハンド・アクション実行
bool Action(int id,uint32_t position,uint32_t effort,uint32_t runTime,
bool wait = false);
指定された id のハンドに対して effort で指定された力（最大）により position の位置へ
runTime で指定された時間で開閉します。
アクションの実行に失敗した場合は false が返ります。

id　ハンドの ID(1～31)
position　開閉位置(0:全開～1000:全閉)
effort　出力 (5:最小～1000:最大)
runTime　開閉時間（~~250:最短~~～10000:最長） ※ミリ秒単位です。
wait　true ： 開閉を完了するまで Action 関数は終了しません。

例 4 sample_4-4-2-2.cpp

・メッセージによる制御 ※ロボットハンドの開閉を繰り返すサンプルコードです。

```
#include  “AspinaHand.h”
int main(int argc, char **argv)
{
  ros::init(argc, argv, "test");
  ros::NodeHandle n;
  ros::Rate loop_rate(1000);
  AspinaHandTopic topic(&n,1);
  int id = 1; // Hand ID = 1
  AspinaHandTopic::STATUS sts;

  if(topic.Startup(id) == false)
  {
    printf("Startup fail.¥n");
    return -1;
  }

  while(ros::ok())
  {
    int ret = topic.GetStatus(id,&sts,false);
    if(ret == 0)
    {
      if(sts.stalled == false && sts.reached_goal == true)
      {
        uint32_t pos;
        if(sts.position < 500)
        {
          pos = 1000;
        }
        else
        {
          pos = 0;
        }
        if(topic.Action(id,pos,1000,250,true) == false)
        {
          return -1;
        }
      }
    }
    else
    {
      if(ret < 0)
      {
        return -1;
      }
    }
    ros::spinOnce();
    loop_rate.sleep();
  }
  return 0;
}
```

以上