ChoreonoidとG-ROBOTを用いたロボットモーション作成

（独）産業技術総合研究所
知能システム研究部門
原　　功

Contents

- Choreonoidの概要
  - –Choreonoidの開発背景
  - –Choreonoidの概要
  - –Choreonoidの使い方
- Choreonoidを用いたロボティクスの研究開発事例

Choreonidの開発背景

- 1996年末　ホンダヒューマノイド　P2発表
- 1998年～2003年　「人間協調・共存型ロボットシステムの研究開発」プロジェクト

以降、さまざまな人型のロボットが登場

→　ロボットの動作教示が複雑化

→　各関節角の目標角度を直接入力では限界がある

→　モーションキャプチャの利用　→ 高価な機器が必要

OpenHRP3などのシミュレータを利用

→ 動作が遅く不安定

→ CGを作るようにもっと簡単に動作を作成できないか？

Choreonidの開発背景

- OpenHRP3のように動力学シミュレーションが実行でき、CG製作者でも簡単に扱えるツールを実現したい

- OpenHRP3の不満点
  - Java3Dを利用しているため、動作が遅い
  - メモリ容量の限界、ガベージコレクション
  - 動力学計算は、C++なのに…

知能化PJにおいて開発したロボット知能ソフトウェアプラットフォームの1つのツールとしてフルスクラッチでから開発　← UIはOpenHRP3との親和性を持たせることが条件で…

基本的な設計方針

- 内部計算処理だけでなく、3Dレンダリングを含む可視化、アニメーション、およびユーザ入力と内部計算との連携も含めて、コンピュータの性能を最大限活用可能な設計とする。
- 必要に応じてユーザが機能を柔軟に拡張可能とする。
- ロボットや計算機の専門家ではないユーザにも使いやすいツールとする。

MVCモデルに基づく実装

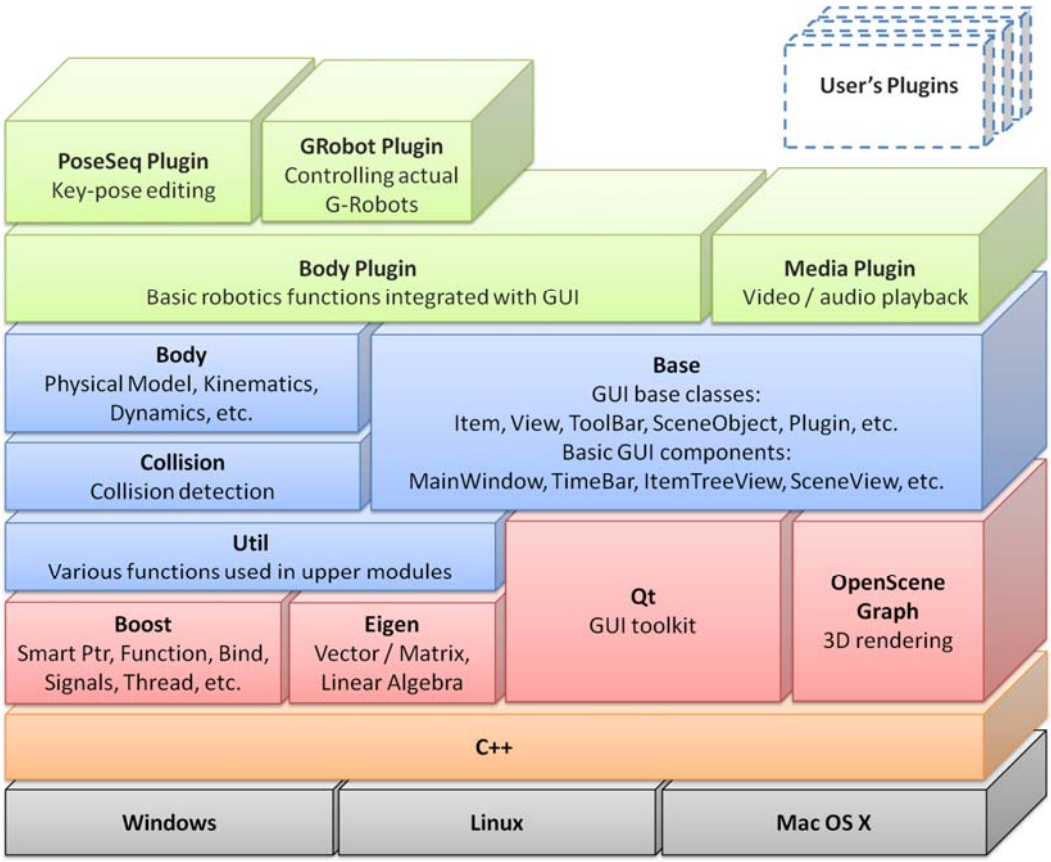
→ロボットアプリケーションのフレームワークとしても利用可能

Choreonidの特徴

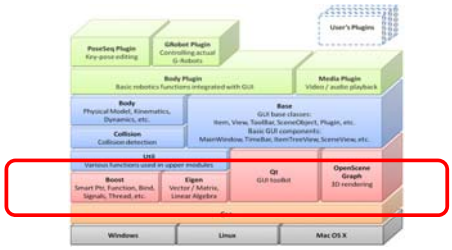
多関節型ロボットの動作パターンを作成するためのGUIツール

- キーフレームベースの姿勢設定と動作補完
  - ユーザは、キーポーズを作成するだけ
- 姿勢設定時に動力学シミュレーションを同時実行
  - 無理な姿勢を自動的に修正
- C++による高速な処理の実現
  - より高速に、より安定に
- プラグインにより様々な機能拡張が可能
  - より柔軟に、拡張可能に

Choreonoidのシステム構成

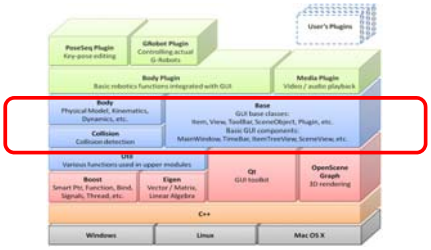


Choreonoidで利用している主な汎用ライブラリ



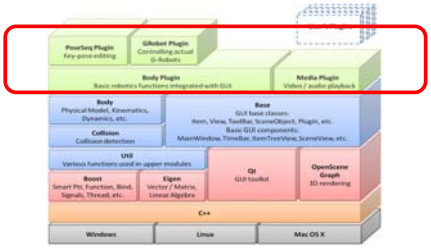
- **Boost C++ Libraries**
  - C++の汎用的なライブラリ集で、標準ライブラリがカバーしない部分をカバーする大変有用なライブラリ集
- **Eigen**
  - 行列・ベクトル演算をはじめとする線形代数処理を扱うC++のテンプレートライブラリ
- **Qt**
  - GUIツールキットライブラリで、多様で高品質なGUI部品を提供する。
- **OpenSceneGraph**
  - 3次元CG描画をシーングラフというハイレベルなAPIで行うためのライブラリ。

Choreonoidの基盤モジュール群



- **Util**: ツール実装の各所で使われるクラスや関数をまとめたユーティリティライブラリ
- **Collision**: 干渉チェック機能を提供するライブラリ
- **Body**: ロボットモデルの定義や各種計算処理を扱うライブラリ
- **Base:**ツールのGUIに関連する基盤機能をまとめたモジュール。

Choreonoidの標準プラグイン



- **BodyPlugin:**ロボットモデルや動作データに関するGUIや処理を実装したプラグイン
- **PoseSeqPlugin:** ロボットのキーフレーム編集を行う機能を実装したプラグイン
- **GRobotPlugin:** HPIジャパン株式会社の小型ロボット”G-Robot”の実機をモデルの動作と連動させて動かすための機能を実装したプラグイン
- **MediaPlugin:** ビデオや音声をロボットの動きと合わせて再生するためのプラグイン

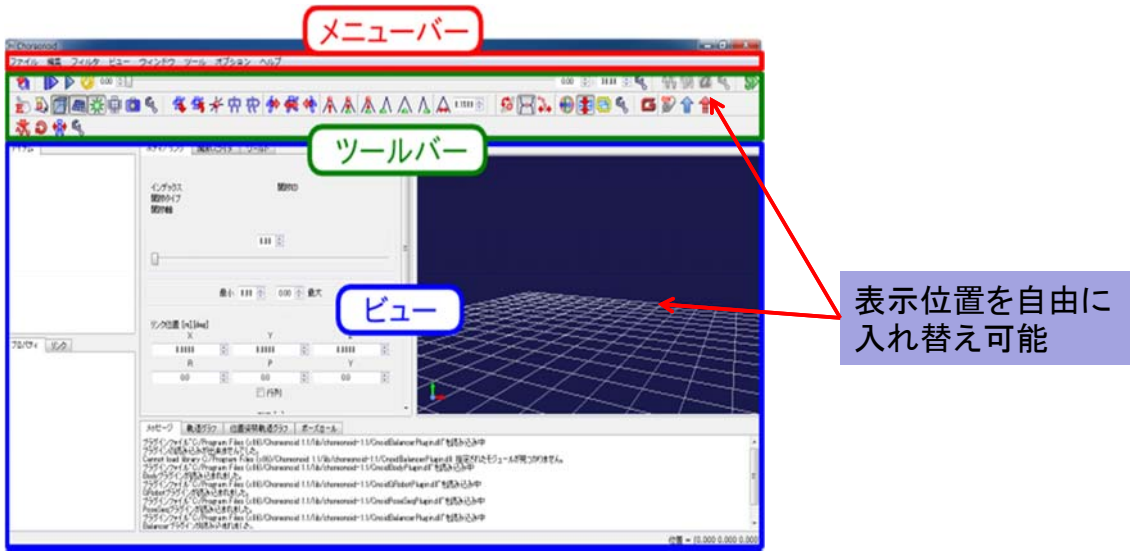
Choreonoidの機能拡張

Choreonoid は、プラグインの追加によって…

– 動作計画ツールとして
– ロボット操作IFとして
– ロボットシミュレータとして
– シナリオ記述するツールとして
– モデルデザインツールとして

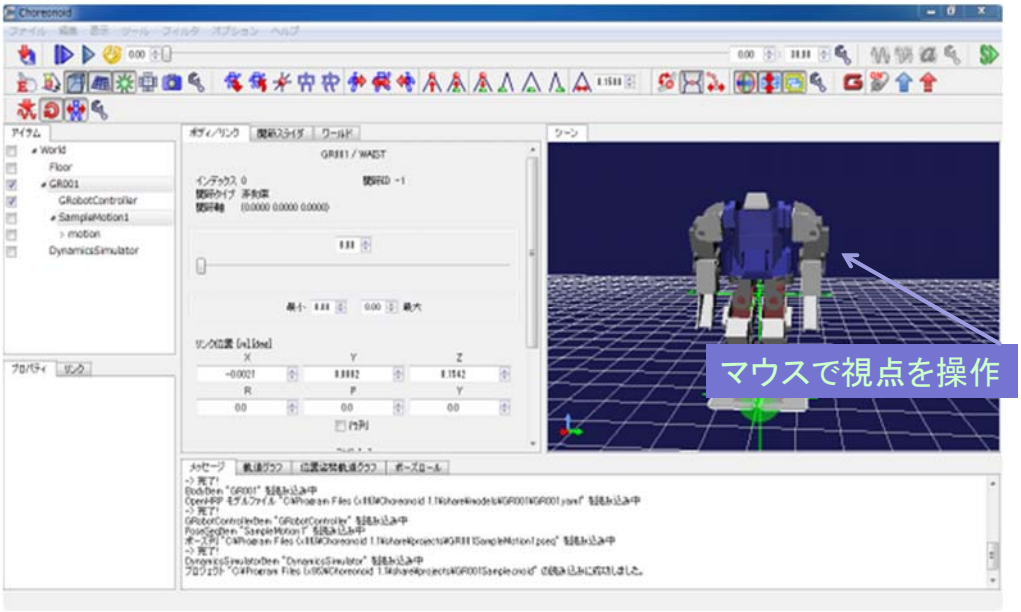
利用することができる

Choreonoidの基本的な操作

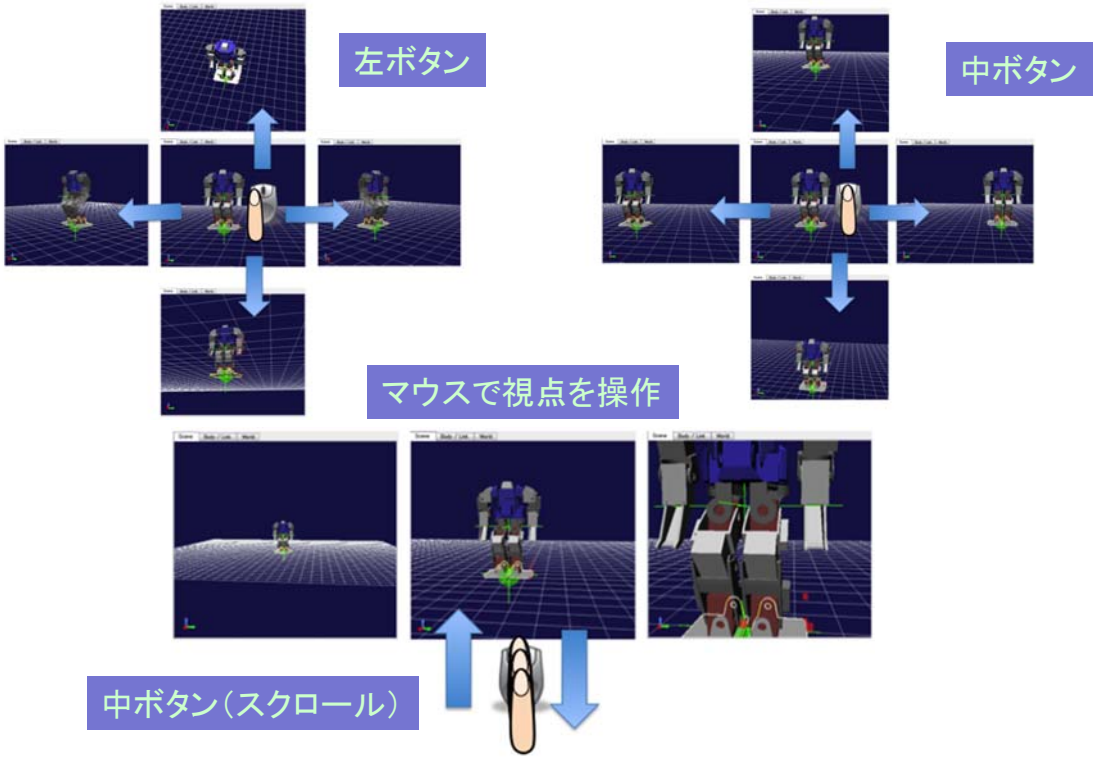


- メニューバー
  - 本メニューバーに格納されているメニューを用いることで、Choreonoidの各種操作を行うことが出来ます。メニュー項目はプラグインによって追加することも可能です。
- ツールバー
  - ツールバー領域には、ボタンやスライダ、数値入力ボックス等のGUI部品で構成されるツールバーが配置されます。ツールバーは機能ごとにグループ化されたものとなっており、各ツールバーの左端をマウスでドラッグすることで簡単に好みの場所に移動させることができます。
- ビュー
  - ビューは、Choreonoidにおいて各種情報を表示・編集するためのウィンドウ領域です。タブ内に格納される各矩形領域がひとつのビューに対応します。
  - Choreonoid本体に備わった基本的なビューとして、各種データ・モデル等（アイテム）を管理する「アイテムビュー」、各アイテムのプロパティを表示・編集するための「プロパティビュー」、3D表示にてモデルの確認や編集を行うための「シーンビュー」、テキストメッセージが出力される「メッセージビュー」などがあります。

Choreonoidの基本的な操作

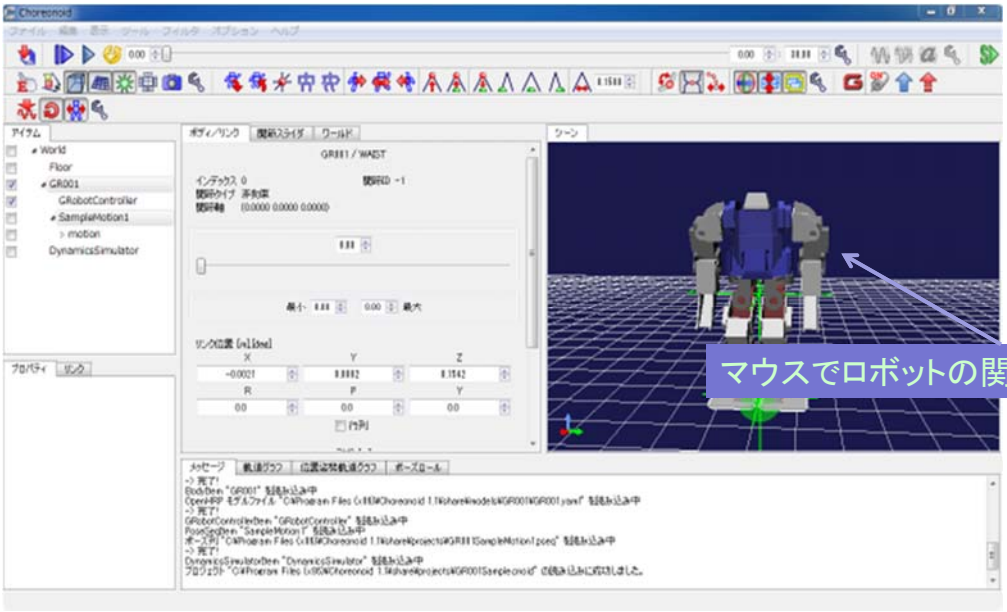


Choreonoidの基本的な操作

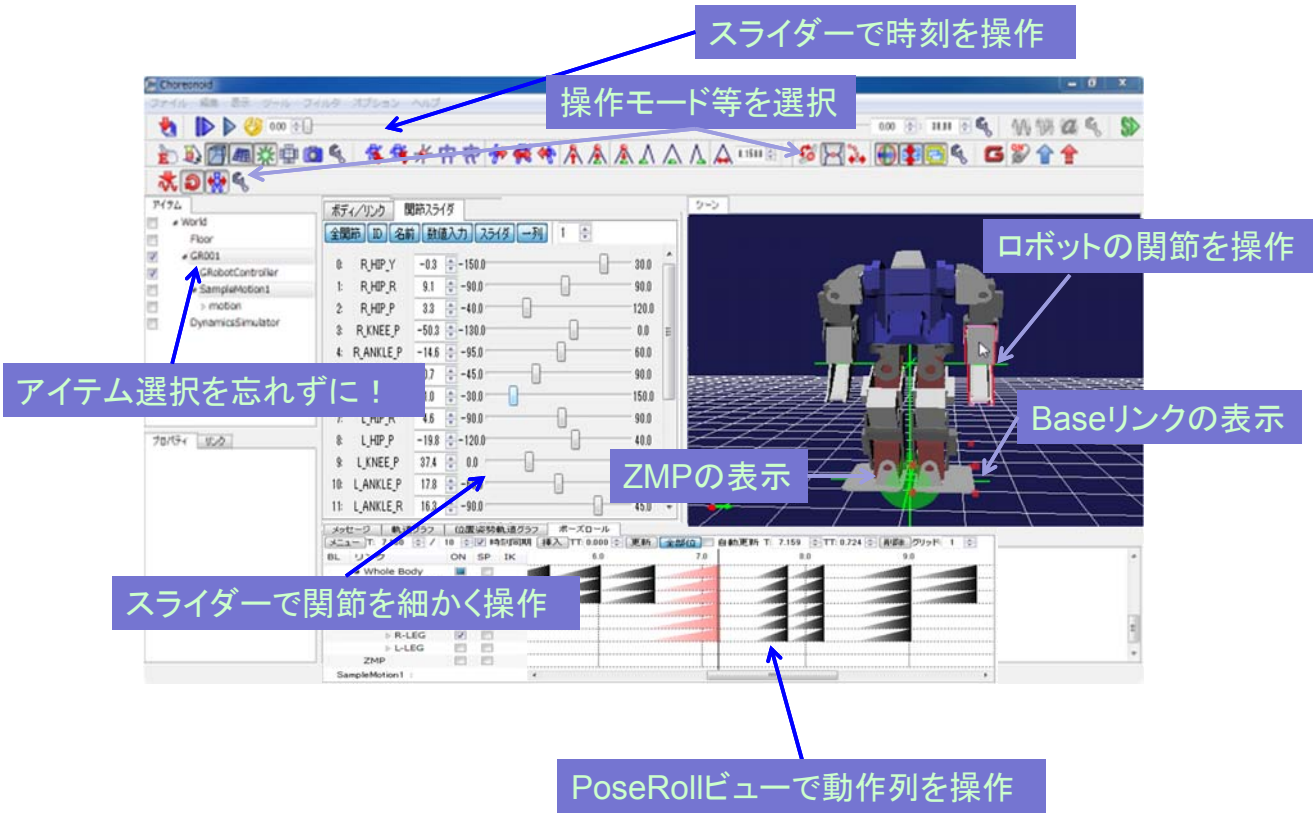


NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST)

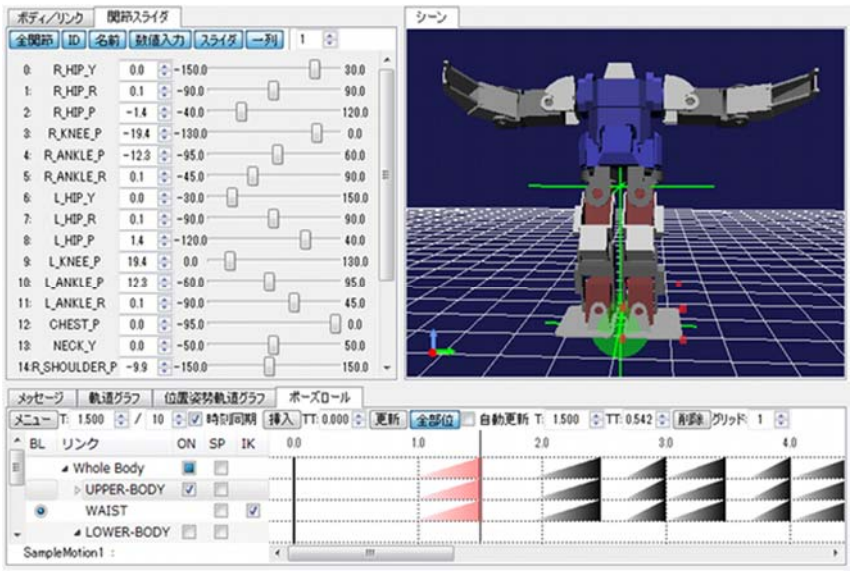
Choreonoidの基本的な操作



Choreonoidの基本的な操作



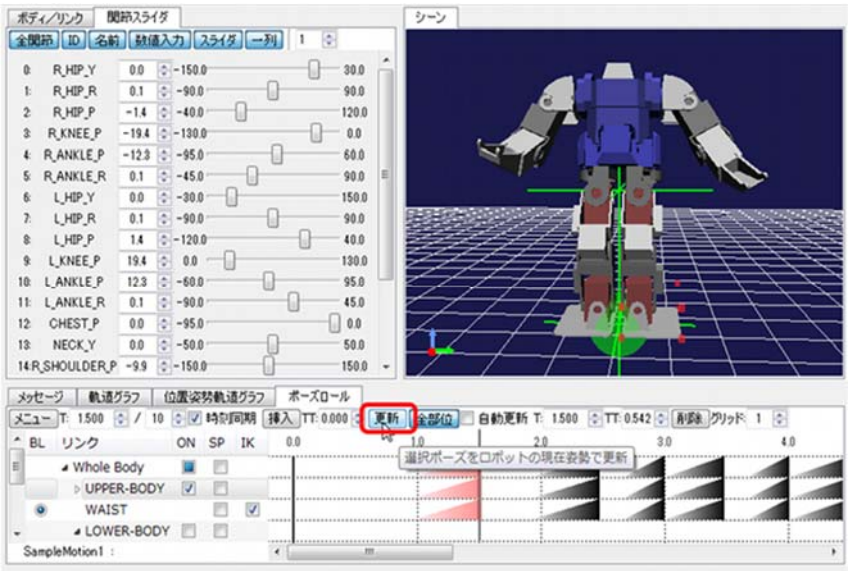
キーポーズの変更と追加



キーポーズの選択

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

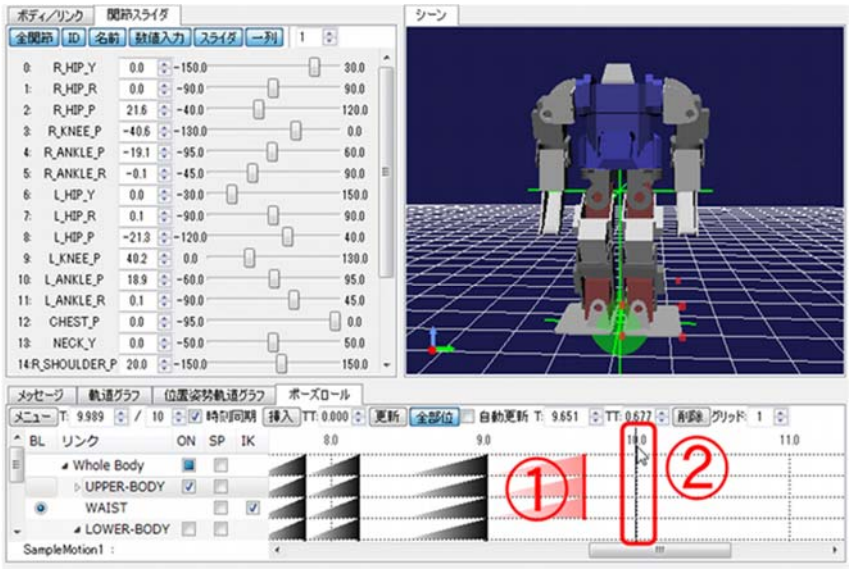
キーポーズの変更と追加



キーポーズを更新

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

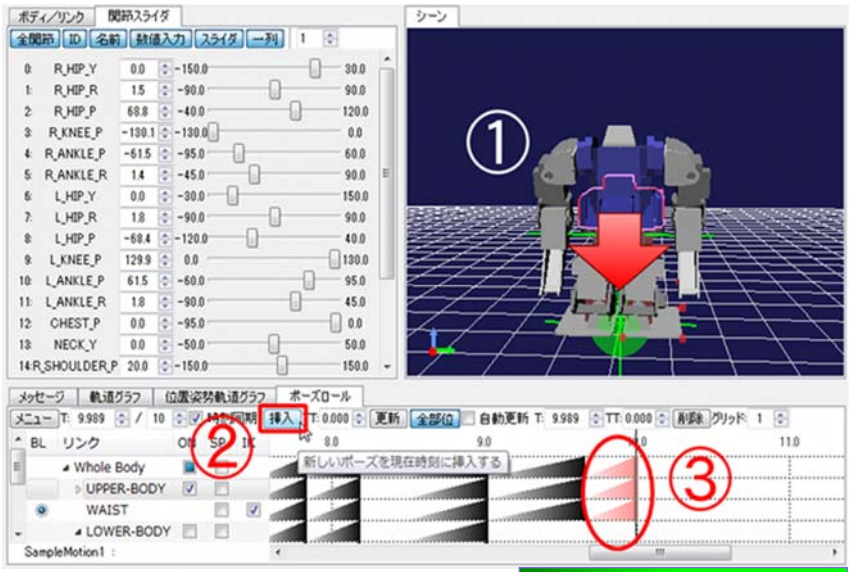
キーポーズの変更と追加



1. 動作の末尾に移動
2. キーポーズの挿入時刻を指定

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

キーポーズの変更と追加



1. キーポーズを生成
2. キーポースを挿入
3. 動作時間を調整

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST)

Choreonoidにおけるロボットの動作表現

- Choreonoidでは、ロボットの動きをキーポーズの連続として表現する
- キーポーズは、動作中の状態が変化するポイントの姿勢
- 無理な姿勢のキーポーズは、身体バランス補正を行い、安定動作を生成する。
- ロボットへは、制御時間ごとの目標姿勢に変換して、命令列を与えて実行させる

Choreonoidで生成される動作パターンファイル

Choreonoidでは、YAML形式のファイルを使用

- Pose Sequenceファイル(.pseq)
  - 時刻、動作時間、動作する関節角の目標角度、IK計算のための情報
- Motionファイル(.yaml)
  - Pose Sequenceから生成される
  - 一定時間間隔の関節角の列、関節重心位置、ZMPの位置などの情報

SampleMotion1.pseq

```
type: PoseSeq
name: "SampleMotion1"
targetBody: "GR001"
refs:
 -
  time: 0
  refer:
   type: Pose
   name: ""
   joints: [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19 ]
   q: [
    -1.80616593e-19, 0.00197016157, 0.370920524, -0.701176708, -0.330256184, 0.00197016157,
    -6.05607271e-19, 0.00197016157, -0.370920524, 0.701176708, 0.330256184, 0.00197016157,
    0.34906585, 0, -0.34906585, -0.34906585, 0, 0.34906585 ]
   ikLinks:
    -
     name: WAIST
     index: 0
     isBaseLink: true
     translation: [ -0.00206589105, 0.000206960048, 0.154197111 ]
     rotation: [
      1, 7.25663958e-19, 7.30074269e-16,
      -4.21994409e-19, 1, 3.56094663e-15,
      -5.75440303e-16, -3.56007164e-15, 1 ]
    - . . .
```

データのタイプ

対象となる関節のID

IKに関する情報

目標関節角

SampleMotion1.yaml

```
# Body motion data set format version 1.0 defined by cnoid-Robotics
type: BodyMotion
components:
 -
  type: "MultiValueSeq"
  purpose: "JointPosition"
  frameRate: 50
  numParts: 20
  numFrames: 534
  frames:
   - [ 5.67373576e-011, 0.00197014097, 0.37092773, -0.701189657, -0.330261928,
     0.00197014107, -8.17846501e-011, 0.00197018217, -0.370927729, 0.701189655,
     0.330261927, 0.00197018206, 0, 0, 0.34906585, 0, -0.34906585, -0.34906585, 0,
     0.34906585 ]

   - [ 5.67373566e-011, 0.00197014105, 0.370927731, -0.701189657, -0.330261927,
     0.00197014115, -8.17846502e-011, 0.00197018225, -0.37092773, 0.701189655,
     0.330261926, 0.00197018214, 0, 0, 0.34906585, 0, -0.34906585, -0.34906585, 0,
     0.34906585 ]

          . . .
```

データのタイプ

更新頻度

関節数

目標関節角

Choreonoidの機能拡張

- Choreonoid プラグイン
  - Choreonoid Plugin を追加することで、Choreonoidに様々な機能を追加することができる
  - GUIを伴ったロボットソフトウェア開発、操作環境を簡単に構築

•Boost Signalsライブラリによるイベント処理
•Boost Bindライブラリによるコールバック関数の自動生成

プラグイン開発の流れ

- Choreonidフレームワークのヘッダをインクルード
- プラグインクラスの定義
  - Cnoid::Plugin　のクラスを継承して定義
  - コンストラクタには、プラグイン間の依存関係を’require’ 関数で通知
  - Initialize関数の定義
    - プラグインを初期化、メニュー、ツールバー等の定義
    - 成功すればtrueを返す
  - プラグインの実行関数の定義
- プラグインエントリの定義

HelloWorldPlugin

```
#include <cnoid/Plugin>
#include <cnoid/MenuManager>
#include <cnoid/MessageView>
#include <boost/bind.hpp>

using namespace cnoid;
using namespace boost;

class HelloWorldPlugin : public Plugin
{
public:

  HelloWorldPlugin() : Plugin("HelloWorld")
  {

  }

  virtual bool initialize()
  {
    Action* menuItem = menuManager().setPath("/View").addItem("Hello World");
    menuItem->sigTriggered().connect(bind(&HelloWorldPlugin::onHelloWorldTriggered, this));
    return true;
  }

private:

  void onHelloWorldTriggered()
  {
    MessageView::mainInstance()->putln("Hello World !");
  }
};

CNOID_IMPLEMENT_PLUGIN_ENTRY(HelloWorldPlugin)
```

ヘッダのインクルード

プラグインクラスの定義

メニューの定義

メニューの実行関数

プラグインエントリの定義

- メニューを選択するとメセージを表示する。

HelloWorldPlugin(メニューの作成)

- MenuManagerの取得

```
MenuManager mMgr = menuManager().setPath("/View");
```

- メニューItemの追加

```
Action* menuItem = mMgr.addItem("Hello World");
```

- SignalProxyの取得

```
SignaleProxy<boost::signal<void(void)>> handle
        = menuItem->sigTriggered();
```

- メニューへ関数を結びつける

```
handle.connect(bind(&HelloWorldPlugin::onHelloWorldTriggered, this));
```

- メンバー関数を汎用関数オブジェクトに変換する

```
boost::function<void(void)> func
    =bind(&HelloWorldPlugin::onHelloWorldTriggered, this);
```

SamplePlugin

```
#include <cnoid/Plugin>
#include <cnoid/ItemTreeView>
#include <cnoid/BodyItem>
#include <cnoid/ToolBar>
#include <boost/bind.hpp>

using namespace boost;
using namespace cnoid;

class Sample1Plugin : public Plugin
{
public:

  Sample1Plugin() : Plugin("Sample1")
  {
    require("Body");
  }

  virtual bool initialize()
  {
    ToolBar* bar = new ToolBar("Sample1");
    bar->addButton("Increment")
       ->sigClicked().connect(bind(&Sample1Plugin::onButtonClicked, this, +0.04));
    bar->addButton("Decrement")
       ->sigClicked().connect(bind(&Sample1Plugin::onButtonClicked, this, -0.04));
    addToolBar(bar);

    return true;
  }

  void onButtonClicked(double dq)
  {
    ItemList<BodyItem> bodyItems =
      ItemTreeView::mainInstance()->selectedItems<BodyItem>();

    for(size_t i=0; i < bodyItems.size(); ++i){
      BodyPtr body = bodyItems[i]->body();
      for(int j=0; j < body->numJoints(); ++j){
        body->joint(j)->q += dq;
      }
      bodyItems[i]->notifyKinematicStateChange(true);
    }
  }
};

CNOID_IMPLEMENT_PLUGIN_ENTRY(Sample1Plugin)
```

ヘッダのインクルード

プラグインクラスの定義

プラグインの依存関係の通知

ツールバーの定義

BodyItemの取得

ツールバーの実行関数

状態変更の通知

プラグインエントリの定義

- 選択されているロボットの姿勢を変更するプラグイン
- ツールバーで操作

SamplePlugin

```
class SamplePlugin : public Plugin {
public:
    SamplePlugin() : Plugin("Sample") { require("Body"); }
    virtual bool initialize() {
        ToolBar* bar = new ToolBar("Sample1");
Increment   bar->addButton("Increment ")
                ->sigClicked().connect(bind(&SamplePlugin::onButtonClicked, this, +0.04));
Decrement   bar->addButton("Decrement ")
                ->sigClicked().connect(bind(&SamplePlugin::onButtonClicked, this, -0.04));
        addToolBar(bar);
        return true;
    }
    void onButtonClicked(double dq) {
      ItemList<BodyItem> bodyItems =
        ItemTreeView::mainInstance()->selectedItems<BodyItem>();
      for(size_t i=0; i < bodyItems.size(); ++i) {
        BodyPtr body = bodyItems[i]->body();
        for(int j=0; j < body->numJoints(); ++j) {
            body->joint(j)->q += dq;
        }
        bodyItems[i]->notifyKinematicStateChange(true);
      }
    }
```

clicked

signal

signal

SamplePlugin(ツールバーの定義)

- ツールバー生成

```
ToolBar* bar = new ToolBar("Sample1")
```

- ボタン生成

```
ToolButton *button = bar->addButton("Increment");
```

- SignalProxyの取得

```
SignalProxy<boost::signal<void(bool)>> click_func = button->sigClicked();
```

- クリック時に呼ばれる関数の結びつけ

```
click_func.connect(bind(&Sample1Plugin::onButtonClicked, this, +0.04));
```

- メンバー関数を汎用関数オブジェクトに変換する

```
boost::function<void(void)> func
  = bind(&Sample1Plugin::onButtonClicked, this, +0.04)
```


SamplePlugin(BodyItemの操作)

- ItemViewから選択されたItemの取得

```
ItemList<BodyItem> bodyItems =
      ItemTreeView::mainInstance()->selectedItems<BodyItem>();
```

- 各BodyItemに対して操作する

```
for(size_t i=0; i < bodyItems.size(); ++i){
    //  スマートポインタに変換する。
    //  typedef boost::shared_ptr<Body> BodyPtr
    BodyPtr body = bodyItems[i]->body();
    //  Bodyオブジェクトに対して処理を実行
    for(int j=0; j < body->numJoints(); ++j){
      body->joint(j)->q += dq;
    }
   // モデル全体とGUIに状態変更を通知
     bodyItems[i]->notifyKinematicStateChange(true);

}
```

プラグインのコンパイルとインストール

- プラグインのコンパイル
  - インストール済みのChoreonoidを使う
    - Choreonoid依存ライブラリ、ヘッダーのパスを適切に設定する必要がある
  - Choreonoid本体のコンパイル環境を使う
    - Choreonoidのソースツリーのextpluginの下にコピーして一括コンパイル
    - CMakeList.txtを書く必要がある
- プラグインのインストール
  - {ChoreonoidのTopDir}/lib/choreonoid-1.1の下にDLLをコピーする

Chreonoidを自分のロボットで利用するには

- Chreonoidを利用するに必要なもの
  - 対象となるロボットのモデル
    - ロボットのパーツのVRMLモデル
    - 各パーツの重心位置と慣性モーメントマトリックス
  - IK計算用プログラムモジュール
    - 独自開発
    - OpenRAVE提供のikfastを利用する
  - ロボット制御ソフトへ命令を与えるためのプラグイン

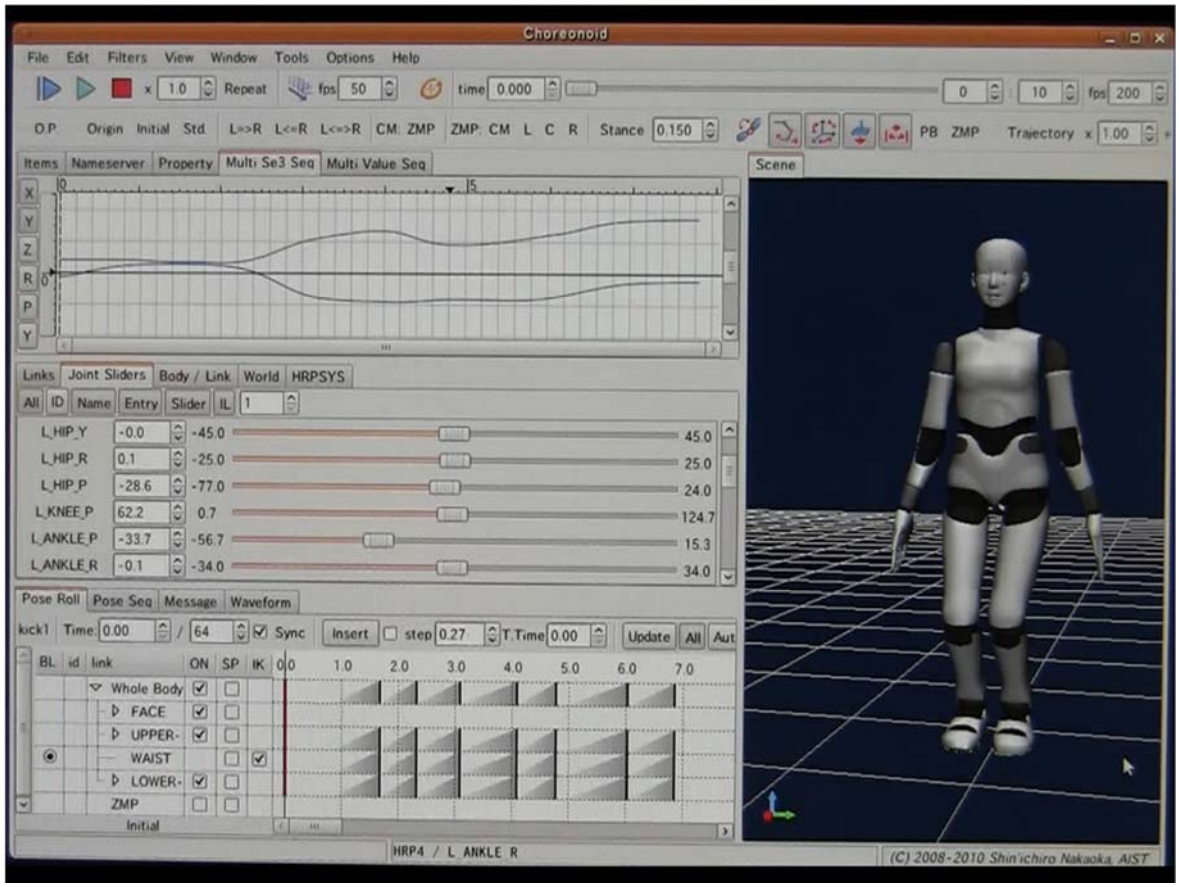
Choreonoid 1.2で追加された機能

- 動力学シミュレーションプラグイン
  - AISTの動力学エンジン（OpenHRP3のもの）
  - ODE (OpenDynamics Engine)
- CORBAへの対応
  - ネームサーバー
  - OpenRTM
- 非CORBA通信コントローラサンプル
- 2Dシミュレーション

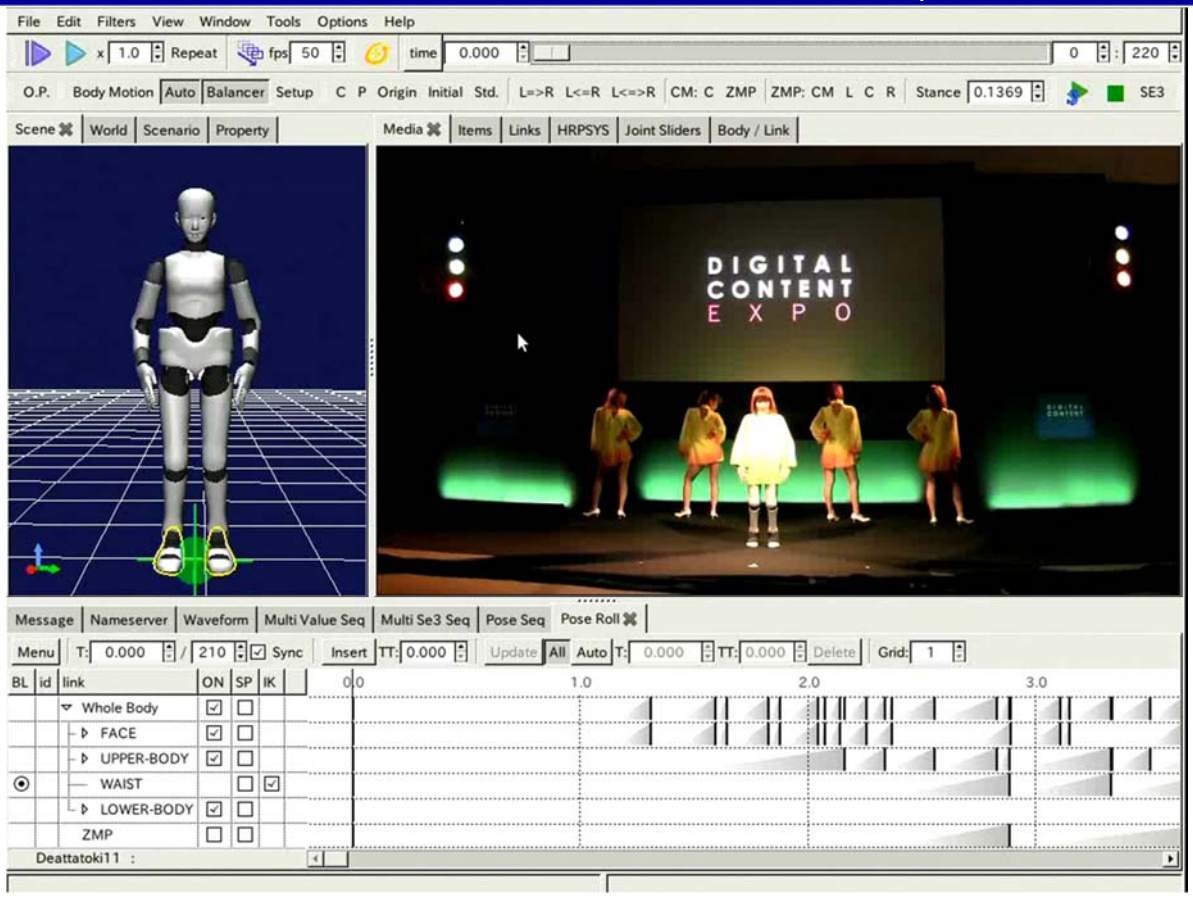
Choreonoidを用いた研究開発事例

- HRP-4C
  - HRP-4C PressRelease
  - CEATEC
  - 国際ロボット展2011
- HRP-4
  - Press Release

Choreomoidの活用事例（HRP-4C）

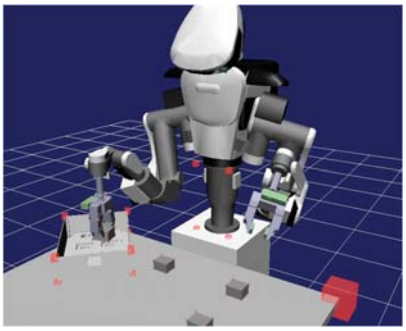


Choreonoidの活用事例（HRP-4C）



Choreonoidの利用例

- 知能化PJ　頭部ステレオカメラを用いた双腕ロボットによるマニピュレーション作業
  - graspPlugin for Choreonoid
  - http://choreonoid.org/GraspPlugin/

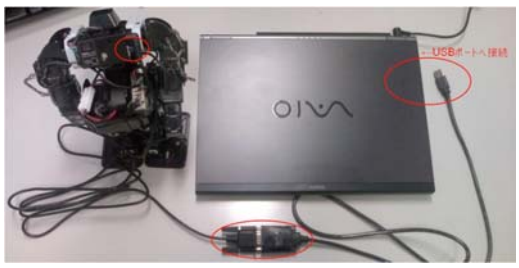


OpenHRIを用いた応用例：ロボット音声IF

- 市販のホビーロボットを音声命令で動作させる
- 音声対話モジュールSEATの状態遷移モデル作成の例
- ロボットの状態（姿勢）に応じて、音声コマンドとの対応を変化させる

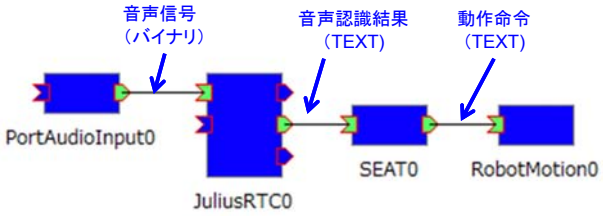
【ハードウェア】

- Windows 7の動作するパソコン
- KINECT
- G-ROBOTS GR-001　または Choreonoid



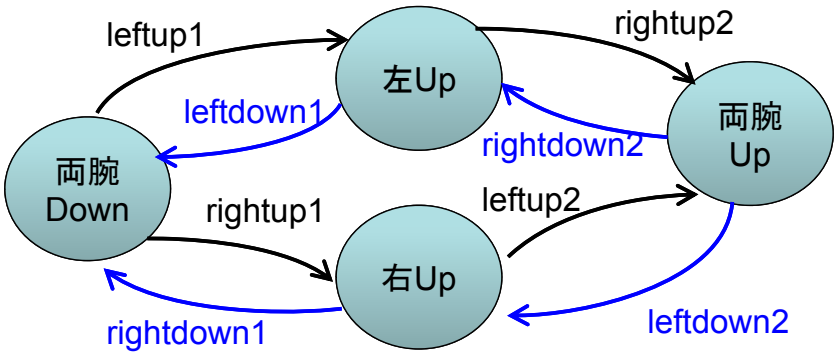
【利用コンポーネント】

•KINECTコンポーネント: 音声データの取得
•Juliusコンポーネント: 日本語音声認識
•SEATコンポーネント : 音声対話制御
•Choreonoid: GR-001シミュレーション



利用するロボットの動作パターン

- Choreonoidで作成した8つの動作
  - leftup1：両腕をおろした状態から左腕を挙げる
  - leftup2:右腕を挙げた状態から左腕を挙げる
  - rightup1：両腕をおろした状態から右腕を挙げる
  - rightup2:左腕を挙げた状態から右腕を挙げる
  - Leftdown1：左腕のみを挙げた状態から左腕をおろす
  - leftdown2:両腕を挙げた状態から左腕を下ろす
  - Rightdown1：右腕のみを挙げた状態から右腕をおろす
  - rightdown2:両腕を挙げた状態から右腕をおろす

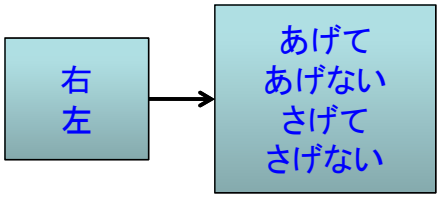


音声認識文法

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-grammar/grammar.xsd"
        xml:lang="jp"
        version="1.0" mode="voice" root="command">

  <rule id="command">
    <one-of>
      <item>右</item>
      <item>左</item>
    </one-of>
    <one-of>
      <item>あげて</item>
      <item>あげない</item>
      <item>さげて</item>
      <item>さげない</item>
    </one-of>
  </rule>

</grammar>
```



音声対話定義

```
<?xml version="1.0" encoding="UTF-8"?>
<seatml>
 <general name="flaggame">
  <agent name="speechin" type="rtcin" datatype="TimedString" />
  <agent name="command" type="rtcout" datatype="TimedString" />
 </general>
 <state name="both_down">
  <rule>
   <key>右 (あげて|さげない)</key>
   <command host="command">rightup1</command>
   <statetransition>right_up</statetransition>
  </rule>
  <rule>
   <key>左 (あげて|さげない)</key>
   <command host="command">leftup1</command>
   <statetransition>left_up</statetransition>
  </rule>
      ＜中略＞
</state>
</seatml>
```


おわりに

- Choreonoidの概要とシステム事例の紹介
  - 動力学シミュレーションを行いながら多関節型ロボットの動作を簡単に記述
  - 直感的で簡易なインターフェースを提供
  - Choreonoid 1.2の情報
  - Choreonoidを使った活用事例

ソフトウェアの動作確認　1

- 準備
  - KINECTを使う場合には、KINECT　SDK 1.5 をインストール
    - KINECT　SDKは、Windows7が必要なので確認
    - USBメモリのKinectSDK-v1.5-Setup.exeを実行し、指示に従いインストールする
  - OpenRTM-aist-1.1.0-Releaseをインストール
    - USBメモリ内のdownloadのフォルダにOpenRTM-aist-1.1.0-RELEASE_vc10.msiで導入する。
  - USBシリアルケーブルを接続して、ドライバをインストール
    - USBメモリの　¥USB-Serial Driver 内にある。
    - デバイスマネージャーでシリアルポートがあることを確認。
  - Choreonoid、OpenHRIなどをインストール
    - 通常は、各ソフトウェアのインストーラで行うが、今回はUSBメモリから導入する
    - USBメモリのSeminar-OpenHRIを C:¥にコピー
    - Seminar-OpenHRI には、Choreonoid1.1, OpenHRI等の必要なモジュールを収録済み

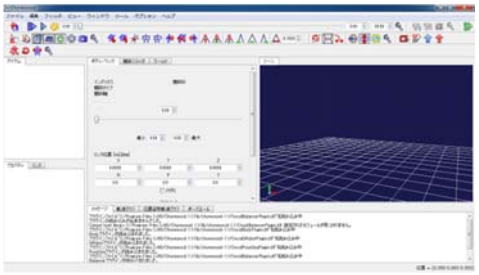
ソフトウェアの動作確認　1

- 準備
  - KINECTを使う場合には、KINECT　SDK 1.5 をインストール
    - KINECT　SDKは、Windows7が必要なので確認
    - USBメモリのKinectSDK-v1.5-Setup.exeを実行し、指示に従いインストールする
  - OpenRTM-aist-1.1.0-Releaseをインストール
    - USBメモリ内のdownloadのフォルダにOpenRTM-aist-1.1.0-RELEASE_vc10.msiで導入する。
  - USBシリアルケーブルを接続して、ドライバをインストール
    - USBメモリの　¥USB-Serial Driver 内にある。
    - デバイスマネージャーでシリアルポートがあることを確認。
  - Choreonoid、OpenHRIなどをインストール
    - 通常は、各ソフトウェアのインストーラで行うが、今回はUSBメモリから導入する
    - USBメモリのSeminar-OpenHRIを C:¥にコピー
    - Seminar-OpenHRI には、Choreonoid1.1, OpenHRI等の必要なモジュールを収録済み

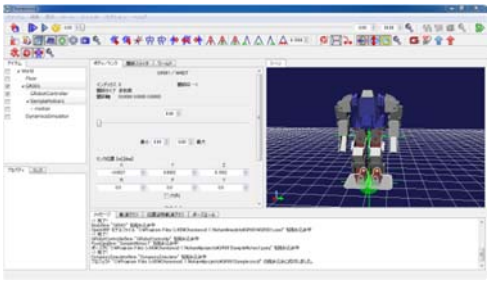
NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST)

ソフトウェアの動作確認　2

- Chorenoidを起動してみる
  - C:¥Seminar-OpenHRIの下にある choreonoid-1.1のショートカットから Choreonoidの起動を確認する。
  - 次に、G-ROBOTのサンプルプロジェクトを読み込む
    - C:¥Seminar-OpenHRI¥Choreonoid-1.1¥share¥projects¥GR001Sample.cnoid
  - サンプル動作の実行
  - G-ROBOT GR001を接続して動作することを確認する



起動直後

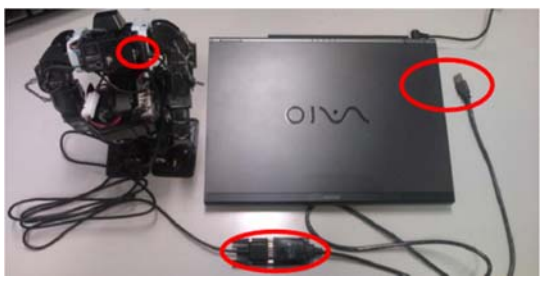


サンプルプロジェクト読み込み後

ソフトウェアの動作確認　3

- G-ROBOT GR001を接続して動作することを確認する

①接続



②ポートの設定

←ここを修正

③接続確認

↑ここのボタンを押してサーボモータのON/OFFを確認

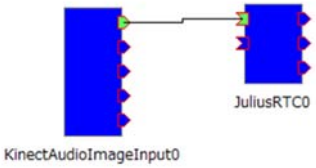
ソフトウェアの動作確認　4

- ChorenoidのRobotMotionRtcの動作を確認
  - スタート → OpenRTM-aist 1.1 → C++ → tools → Start Naming Service　でネームサーバーを起動する。
  - C:¥Seminar-OpenHRIの下にある choreonoid-1.1 のショートカットからChoreonoidの起動する。
  - G-ROBOTのプロジェクトを読み込む
    - C:¥Seminar-OpenHRI¥Choreonoid-1.1¥share¥projects¥GR001.cnoid
  - RTシステムエディタを起動し、NameServiceビューに　Cnoid_RobotMotion0|rtc　があることを確認

ソフトウェアの動作確認　5

- KINECTコンポーネントの動作確認
  - スタート → OpenRTM-aist 1.1 → C++ → tools → Start Naming Service　でネームサーバーを起動する。
  - C:¥Seminar-OpenHRIのフォルダ内のKinectRTCというショートカットを起動する。起動完了まで4秒待つ必要がある。
  - スタート → OpenRTM-aist 1.1 → C++ → tools → RTSystemEditor　でRTシステムエディタを起動しKINECTコンポーネントが起動していることを確認する。

- Juliusrtcを起動して、KINECTコンポーネントを接続し、音声認識できていることを確認する。
  - C:¥Seminar-OpenHRIのフォルダ内のjuliusrtcというショートカットでコンポーネントを起動する。文法ファイルの選択ダイアログがでてきたら、C:¥Seminar-OpenHRI¥Sample内のsimple_demo.grxmlを選択。
  - RTシステムエディタで起
    「さようなら」、「こんにち



続し、アクティベートして「ばいばい」、
ことをコンソールで確認。