次世代ロボット知能化技術開発プロジェクト

冗長性利用モジュール
仕様書・取扱説明書

2011 年 7 月 8 日

国立大学法人　東北大学

株式会社　パイケーク

更新履歴

改定日付	
2009/12/14	新規作成
2010/05/17	4,6章 ファイル名・ディレクトリ記述を修正
2011/03/28	OpenRTM 1.0対応
2011/07/08	インストール手順、終了コマンドの修正

目次

１．はじめに

冗長性利用モジュールは次の機能を実装した RT コンポーネントです。

- 目標手先位置が入力されると、 その手先位置に応じたロボットアームの関節角度を計算し、 目標関節角度を出力
- 現在のロボットの関節角度の状態から現在の手先位置(x,y,z,r,p,y)を求めて出力

図 1 は、冗長性利用モジュールと他のモジュールとの接続関係を表した概要です。

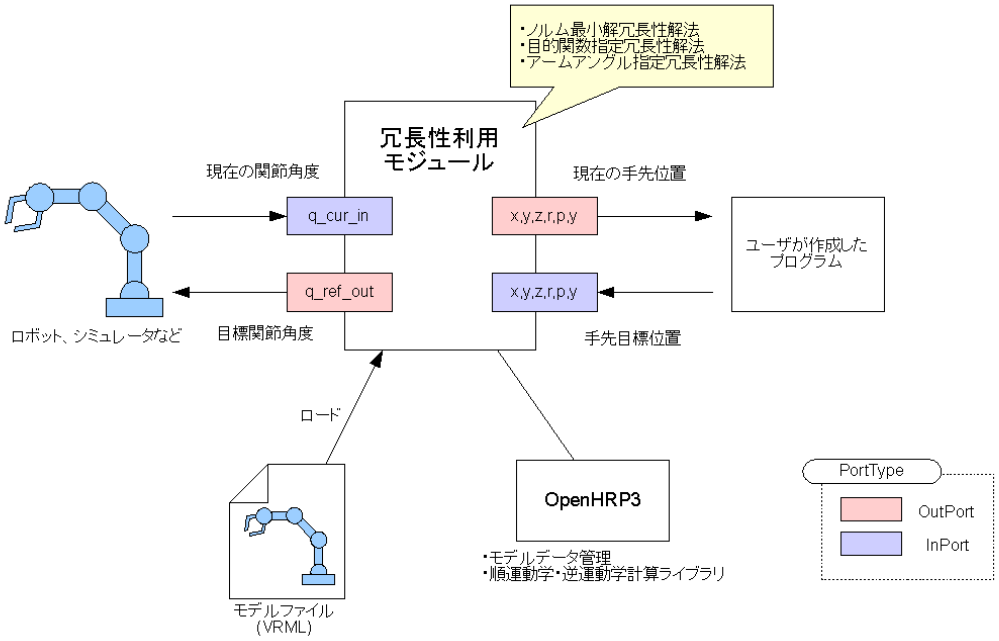

図 1 冗長性利用モジュール接続概要

冗長性利用モジュールは、下記の解法をそれぞれ実装した 3 種類のモジュールを用意しています。

- (1) ノルム最小解冗長性解法モジュール
- (2) 目的関数指定冗長性解法モジュール
- (3) アームアングル指定冗長性解法モジュール

この 3 つの冗長性解法モジュールは、実装しているアルゴリズムは異なりますが、入出力ポートのインターフェース仕様をそろえています。目的に応じて簡単にモジュールを交換することが可能です。

冗長性利用モジュールはロボットアームの移動方法については考慮していません。ロボットアームの各関節を目標角度に実際に駆動するのは、サンプルシミュレーションの場合は関節移動モジュール、実機のロボットアームの場合はモータコントローラが行うことを想定しています。

２．冗長性利用モジュール

２．１．データ入出力ポート

ポート名称	入出力	概要
q_cur_in	入力	ロボットアームの現在の関節角度
q_ref_out	出力	計算されたロボットアームの目標関節角度
xyzrpy_in	入力	ロボットアームの手先の目標位置
xyzrpy_out	出力	現在のロボットアームの手先位置

※ アームアングル指定冗長性解法モジュールの場合には、入力として手先位置(x,y,z,r,p,y)以外に、肘角度(phi_in)を指定する必要があります。

２．２．コンフィギュレーションインターフェース

コンフィグレーション名称	データ型	概要
str_model_file	string	使用するモデルファイル名を指定
str_link_base	string	計算対象とするリンクパスの始点をリンク名称で指定
str_link_end	string	計算対象とするリンクパスの終点をリンク名称で指定
double_dt	double	サンプリング周期
double_limmit_per_sec	double	SC 法適応時の 1 秒あたりの最大変化量

関節(link)の数が 6 未満だと、計算ができないため、エラーとなります。SC 法適応時に、シミュレータなどが期待している動作をしないときは、double_limmit_per_sec の値をチューニングすると改善する場合があります。

３．サンプルシミュレーション

冗長性利用モジュールの利用例として、OpenHRP3 に付属の GrxUI を利用したサンプルシミュレーションについて説明します。

３．１．モジュール間の接続について

図 2 はサンプルシミュレーションを行う際のモジュール間の接続の構成図です。

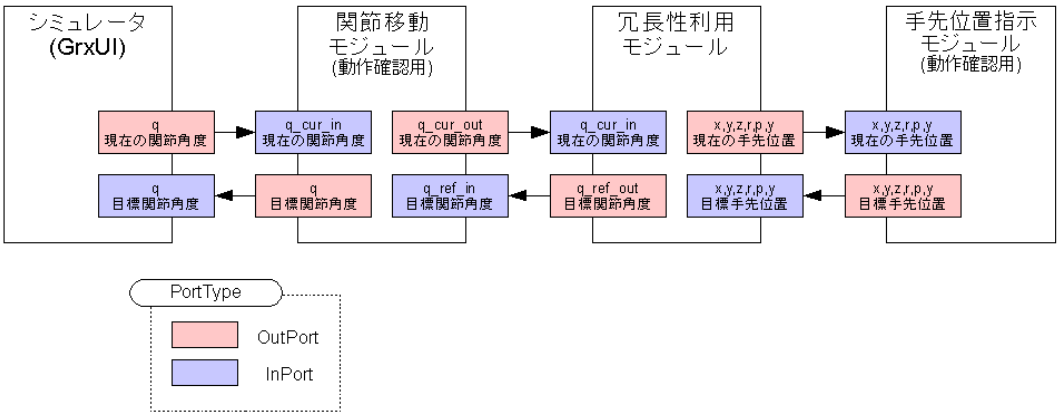

図 2 モジュール間接続構成

３．２．RT System Editor での接続例

以下は RT System Editor 上で関節移動モジュール、冗長性利用モジュール、手先位置指示モジュールを接続した例です。

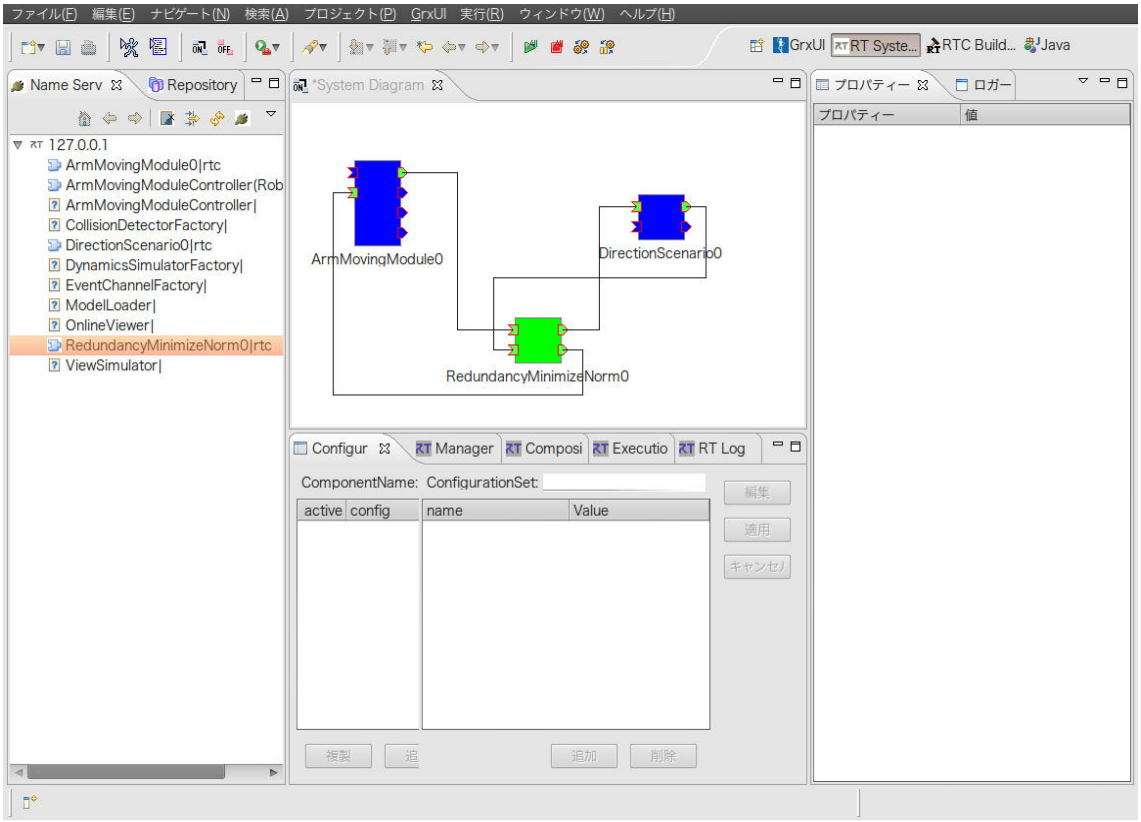

図 3 接続例

シミュレータはシミュレーション開始時に GrxUI から自動的に起動・接続されるので、RT System Editor 上には表示されていません。

３．３．シミュレータ(OpenHRP3)

OpenHRP3 に付属の GrxUI で起動されるシミュレータの本体です。実際には GrxUI から起動される ControllerBridge が接続されます。

サンプルシミュレーションでは、ハイゲインモードを使用して ControllerBridge と接続しています。

３．３．１．入出力ポートの構成

ポート名称	入出力	概要
q	出力	シミュレータ内のロボットアームの現在の関節角度
q	入力	シミュレータ内のロボットアームに設定する関節角度

３．４．関節移動モジュール

サンプルシミュレーション用に作成した RT コンポーネントです。

シミュレータと冗長性利用モジュールは入出力ポートのインターフェース仕様が異なるため直接接続することはできません。そのため、関節移動モジュールはシミュレータと冗長性利用モジュールとのブリッジの役目を果たしています。

関節移動モジュールは現在のロボットアームの関節角度(q_cur_in)から、冗長性利用モジュールから設定された目標関節角度(q_ref_in)へ徐々に近づける動作を行います。実機のロボットアームの場合のモーターコントローラに相当する動作を行っています。

出力 q の値は次のように求めています。

```
diff = q_ref_in - q_cur_in
q = q_cur_in + Kp * diff
```

３．４．１．データ入出力ポートの構成

ポート名称	入出力	概要
q_cur_in	入力	ロボットアームの現在の関節角度
q_cur_out	出力	ロボットアームの現在の関節角度。内容は q_cur_in と同じ。 他のモジュールから参照するために使用
q_ref_in	入力	ロボットアームに設定する目標関節角度
q	出力	ロボットアームに設定する関節角度

３．４．２．コンフィギュレーションインターフェースの構成

コンフィグレーション名称	データ型	概要
double_Kp	double	出力 q の値を求める際に使用する Kp の値

３．５．手先位置指示モジュール

サンプルシミュレーション用に作成した、冗長性利用モジュールに対してロボットアームの手先の目標位置を設定する RT コンポーネントです。

本サイトではシナリオファイル(※)から手先位置を読み込み、冗長性利用モジュールに対して手先の目標位置を設定する RT コンポーネントを用意しています。

- ※ここでのシナリオファイルとは、csv フォーマットなどで時系列で目標位置を記録しているファイルなどのことを指しています。

３．５．１．データ入出力ポートの構成

ポート名称	入出力	概要
xyzrpy_in	入力	現在のロボットアームの手先位置
xyzrpy_out	出力	ロボットアームの手先の目標位置

４．冗長性利用モジュールのインストール

４．１．必要環境

各モジュールは下記の環境で動作確認を行っています。

- Ubuntu 10.04 Desktop Edition
  - http://www.ubuntu.com/
- OpenRTM-aist-1.0.0-RELEASE
  - http://www.openrtm.org/openrtm/node/849
- OpenRTM-aist-Python-1.0.0-RELEASE
  - http://www.openrtm.org/openrtm/node/932
- rtshell
  - http://www.openrtm.org/openrtm/ja/node/869
- OpenHRP3 Ver.3.1.0-Release
  - http://www.openrtp.jp/openhrp3/jp
- Ruby 1.8.7
  - http://ruby-lang.org/ja/
- log4cxx
  - http://logging.apache.org/log4cxx/

Ubuntu 10.04 Desktop Edition をインストール後、必ず

```
$ apt-get update & apt-get dist-upgrade
```

のコマンドを実行し、ライブラリ・プログラムなどを最新状態にしておきます。

OpenRTM 環境のインストールは次の手順で行ってください。

```
$ wget
http://openrtm.org/svnroot/OpenRTM-aist/trunk/OpenRTM-aist/build/pkg_install100_
ubuntu.sh
$ sudo sh pkg_install100_ubuntu.sh
$ wget
http://www.openrtm.org/pub/OpenRTM-aist/python/install_scripts/pkg_install_pytho
n_ubuntu.sh
$ sudo sh pkg_install_python_ubuntu.sh
$ wget http://www.openrtm.org/pub/OpenRTM-aist/tools/1.0.0/rtctree-3.0.0.tar.gz
$ tar xvfz rtctree-3.0.0.tar.gz
$ cd rtctree-3.0.0
$ sudo python setup.py install
$ cd ..
$ wget
http://www.openrtm.org/pub/OpenRTM-aist/tools/1.0.0/rtsprofile-2.0.0.tar.gz
$ tar xvfz rtsprofile-2.0.0.tar.gz
$ cd rtsprofile-2.0.0
$ sudo python setup.py install
$ cd ..
$ wget http://www.openrtm.org/pub/OpenRTM-aist/tools/1.0.0/rtshell-3.0.0.tar.gz
$ tar xvfz rtshell-3.0.0.tar.gz
$ cd rtshell-3.0.0
$ sudo python setup.py install
$ cd ..
```

OpenHRP3 Ver.3.1.0-Release のインストールは次の手順で行います。

```
$ sudo vi /etc/apt/sources.list
  次の行を追加
  deb http://archive.canonical.com/ubuntu lucid partner

$ sudo apt-get update
$ sudo apt-get install openhrp3.1

$ sudo update-alternatives --config java
  ※java-6-sunが選択されていることを確認

$ wget
http://www.openrtp.jp/openhrp3/download/eclipse342_hrpdependencies_linux_ja_2010
0602.tar.gz
$ tar xvfz eclipse342_hrpdependencies_linux_ja_20100602.tar.gz
$ mv eclipse eclipse342_hrpdependencies_linux_ja_20100602
```

```
$ vi eclipse342_hrpdependencies_linux_ja_20100602/eclipse.ini

  次の修正を行う
  --- eclipse.ini.org  2011-03-02
                        18:08:42.275122261 +0900
  +++ eclipse.ini  2011-03-02
                        18:08:50.792122115 +0900
  @@ -7,5 +7,5 @@
   --launcher.XXMaxPermSize
   256m
   -vmargs
  --Xms64m
  --Xmx512m
  +-Xms256m
  +-Xmx1024m

$ cp /usr/share/OpenHRP-3.1/java/plugins/*.jar
eclipse342_hrpdependencies_linux_ja_20100602/plugins/
$ sudo mv eclipse342_hrpdependencies_linux_ja_20100602 /usr/local/
$ cd ~

$ mkdir bin
$ vi ~/bin/eclipse.sh

  次の内容を記述

  #!/bin/sh
  export GDK_NATIVE_WINDOWS=1
  /usr/local/eclipse342_hrpdependencies_linux_ja_20100602/eclipse -vmargs
-Dorg.eclipse.swt.browser.XULRunnerPath=/usr/lib/xulrunner-1.9.2.18/xulrunner
-clean
  ※起動時には~/bin/eclipse.shを使ってEclipseを起動すること。
```

現状の OpenHRP3.1.0-Release には pkg-config の設定にパスが正しく設定されていない問題があるため、OpenHRP3 関係のソースコードをコンパイルを行うことができない不具合があります。正しくコンパイルを行うことができるように /usr/lib/pkgconfig/openhrp3.1.pc ファイルを次のように修正します。

```
prefix=/usr/local
  ↓
prefix=/usr
```

コンパイルに使用する Ruby と log4cxx は、次の手順でインストールを行います。

```
$ sudo apt-get install ruby ruby-dev
$ sudo apt-get install liblog4cxx10 liblog4cxx10-dev
```

４．２．コンパイル手順

各モジュールは*.tar.gz ファイルを取得・展開した後、make コマンドを使用してコンパイルを行います。ファイル名の中の????の部分にはモジュールのリリース日付が入ります。

```
$ tar xvfz redundancy_solution_module-201?????.tar.gz
$ cd redundancy_solution_module-201?????

$ ls
Makefile*                  arm_moving_module-201?????.tar.gz
redundancy_elbow_angle-201?????.tar.gz
PA10.rtc-201?????.tar.gz   build.rb*
redundancy_maximize_manipulability-201?????.tar.gz
README.txt                 direction_scenario-201?????.tar.gz
redundancy_minimize_norm-201?????.tar.gz

$ make extract
  ※各モジュールのtar.gzファイルを展開します。

$ make
  ※各モジュールのコンパイルを行います。
```

５．プログラム実行手順

各モジュールを単体で実行する場合は、ソースコード一式が格納されているディレクトリ内で"make run"を実行すると、モジュールを実行することができます。

下記はノルム最小解冗長性解法モジュールを実行する例です。

```
$ cd redundancy_minimize_norm
$ make run
./RedundancyMinimizeNormComp
getcwd() =
/home/test/work/redundancy_solution_module-20??????/redundancy_minimize_norm
2011-03-28 17:36:14,731 [0x42f8f0e0] INFO
RedundancyMinimizeNorm(RedundancyMinimizeNorm.cpp:99) - onInitialize()
      .
      .
      .
```

５．１．冗長性利用モジュールを単体で起動する際の注意

ノルム最小解冗長性解法モジュール、目的関数指定冗長性解法モジュール、アームアングル指定冗長性解法モジュールを単体で実行する場合は、あらかじめ OpenHRP3 に付属の ModelLoader モジュールを起動しておく必要があります。

冗長性利用モジュールはモデルファイル(*.wrl)のロードに ModelLoader モジュールを利用しているため、ModelLoader モジュールが起動していないとモデルファイルのロードに失敗します。

以下は、ModelLoader モジュールを単体で起動する際の実行例です。

```
$ /usr/bin/openhrp-model-loader
ready
      .
      .
      .
```

あらかじめ GrxUI を起動している場合はこの手順は必要ありません。GrxUI を起動すると GrxUI は ModelLoader モジュールを自動的に起動します。

６．サンプルシミュレーションの実行手順

GrxUI を使ったノルム最小解冗長性解法モジュールのサンプルシミュレーションの操作手順を説明します。図 4 はシミュレーションの実行中の様子です。

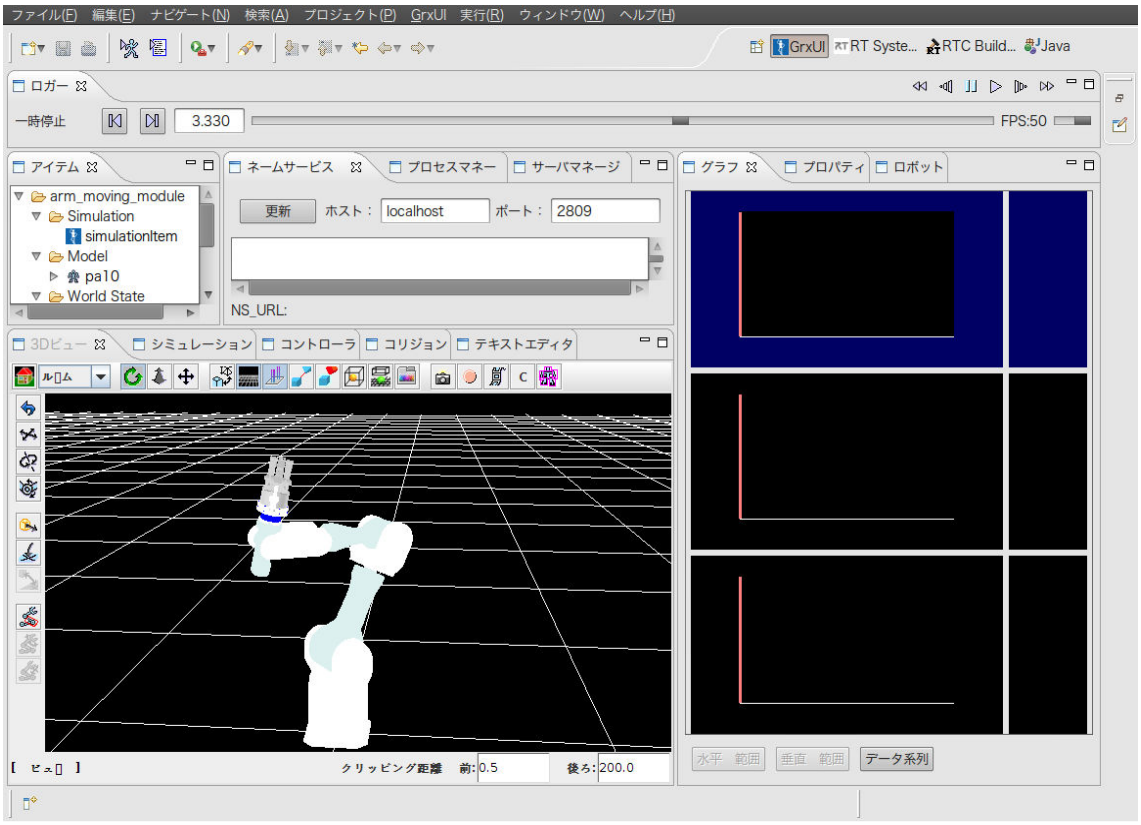

図 4 シミュレーションの実行例

説明では、ホームディレクトリに redundancy_solution_module-20?????.tar.gz が展開されていてコンパイル済みの状態を想定して説明を進めます。

```
$ cd ~
$ tar xvfz redundancy_solution_module-20?????.tar.gz
$ cd redundancy_solution_module-20?????
$ make extract
$ make
```

まずはじめに GrxUI を起動します。

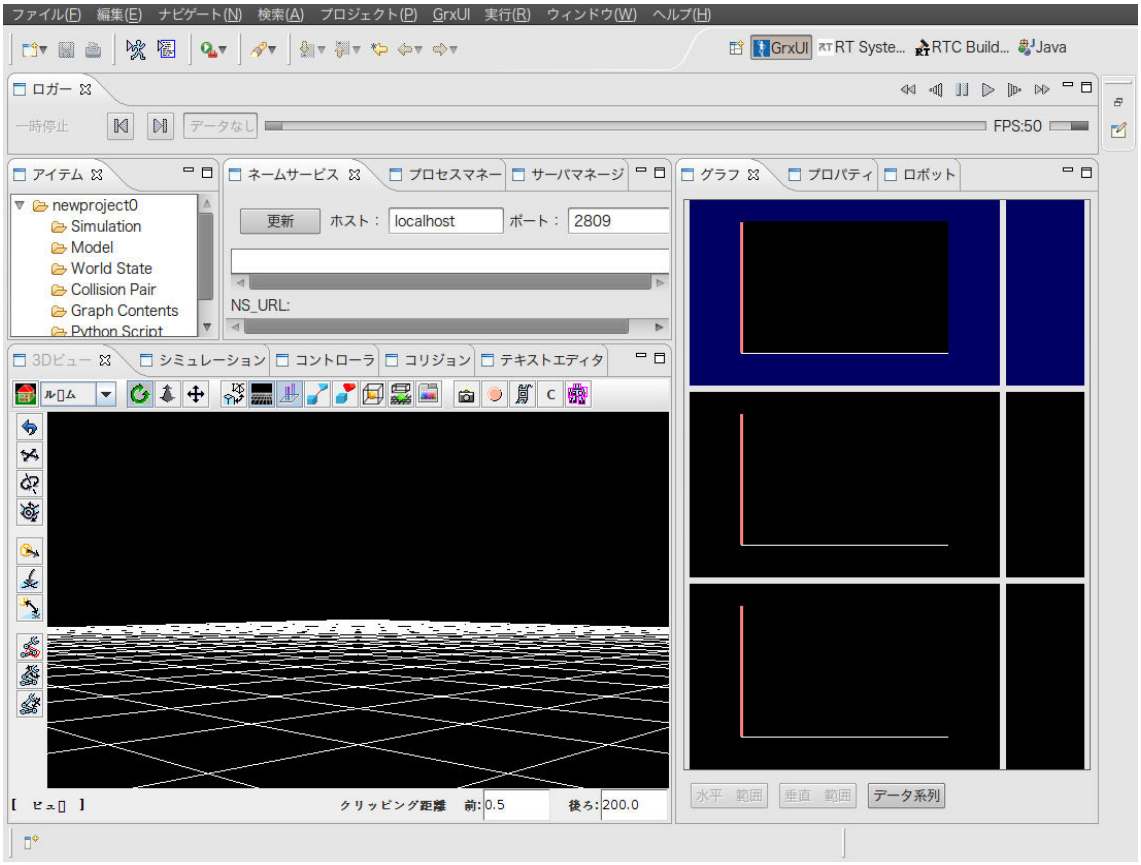

図 5 GrxUI の起動

次に、redundancy_solution_module-20???? ディレクトリ内で、"make setup_redundancy_minimize_norm"を実行します。

```
$ make setup_redundancy_minimize_norm
```

実行すると、ArmMovingModule, ReduncancyMinimizeNorm, DirectionScenario モジュールのポートが接続された状態で起動します。各モジュールは次のように接続されます。

モジュール名	出力ピン	モジュール名	入力ピン	
ArmMovingModule	q_cur_out	ReduncancyMinimizeNorm	q_cur_in	ロボットアームの現在の関節角度
ReduncancyMinimizeNorm	q_ref_out	ArmMovingModule	q_ref_in	ロボットアームの目標関節角度
ReduncancyMinimizeNorm	xyzrpy_out	DirectionScenario	xyzrpy_in	ロボットアームの現在の手先位置
DirectionScenario	xyzrpy_out	ReduncancyMinimizeNorm	xyzrpy_in	ロボットアームの目標手先位置

次に、GrxUI のメニューにある"GrxUI"->"プロジェクトの読み込み"を選択し、arm_moving_module ディレクトリ中にある arm_moving_module.xml ファイルをプロジェクトとして開きます。

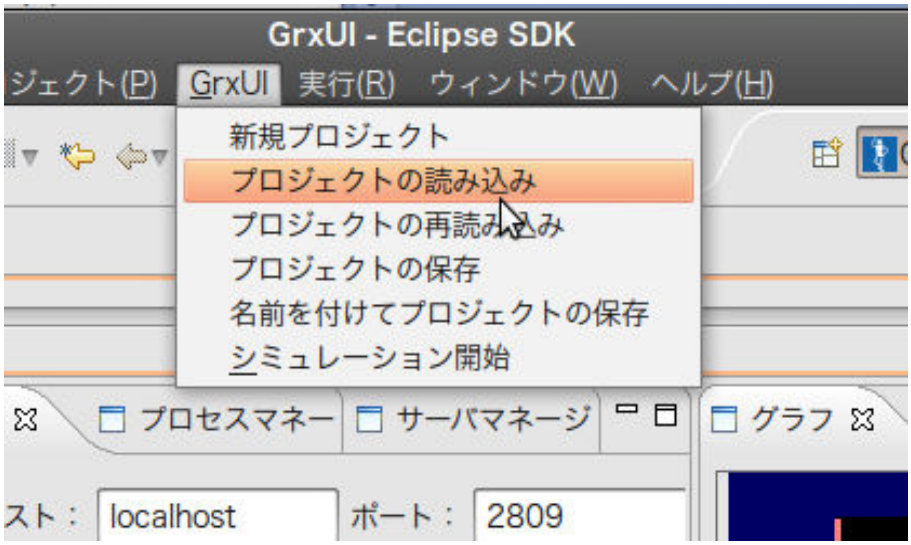

図 6 "プロジェクトの読み込み"メニュー

プロジェクトを開く途中、wrl ファイルの場所を指定するダイアログが表示されます。PA10.rtc ディレクトリの中にある pa10.wrl ファイルを指定してください。

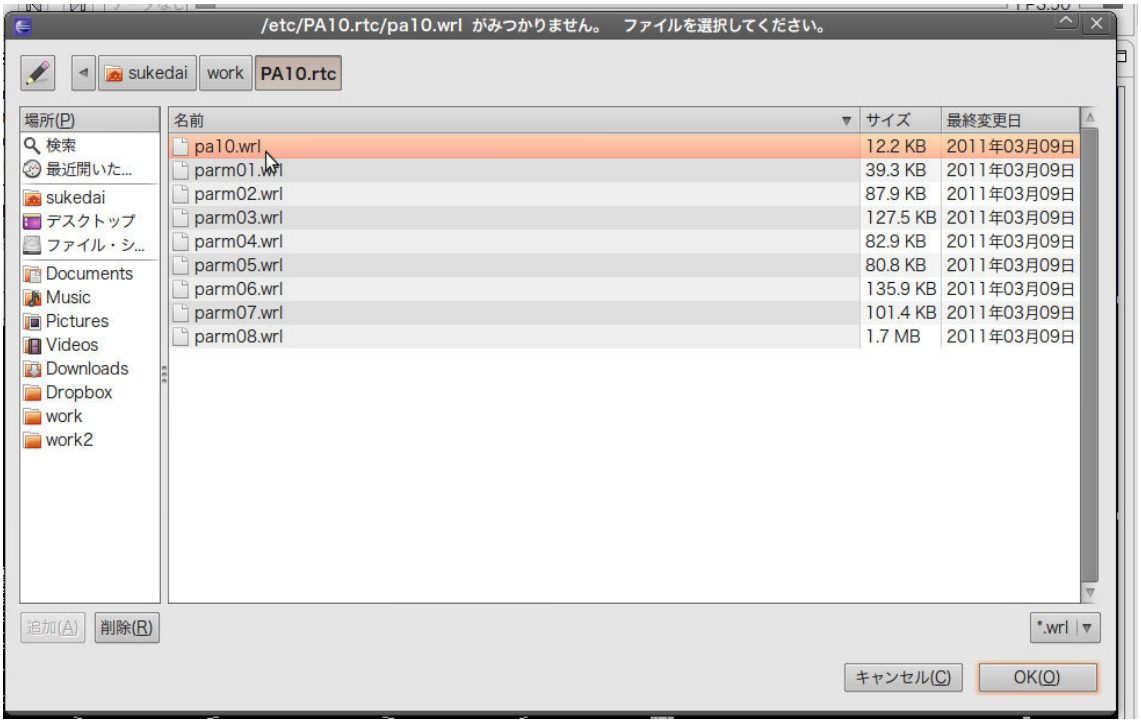

図 7 wrl ファイルの指定

正しくプロジェクトを開くことができると、図 8 のように GrxUI の 3D ビューにモデルが表示された状態になります。

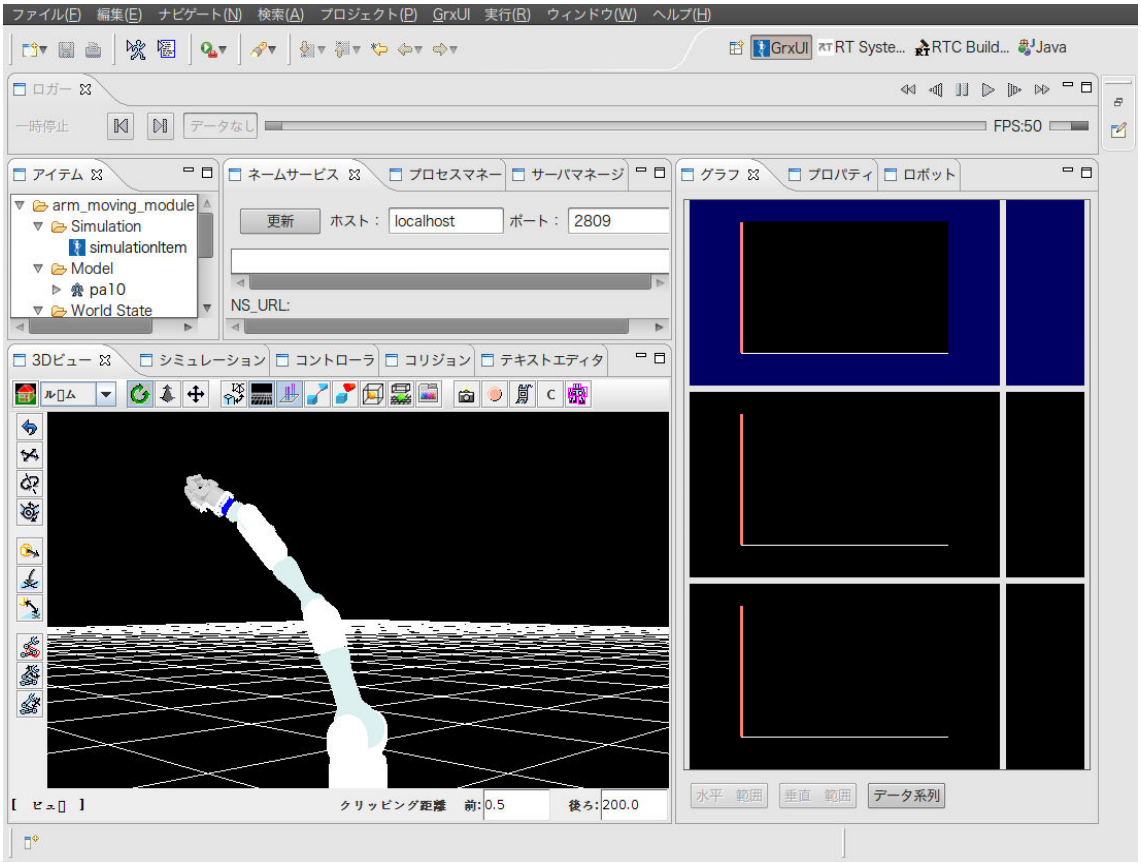

図 8 プロジェクト初期状態

サンプルシミュレーションを開始する場合は、図 9 の GrxUI の"Start Simulation"ボタンを押します。

図 9 "Start Simulation"ボタン

シミュレーションを開始すると、ノルム最小解冗長性解法モジュールを使用して関節角度を求めた結果が図 10 のように GrxUI に順次表示されます。

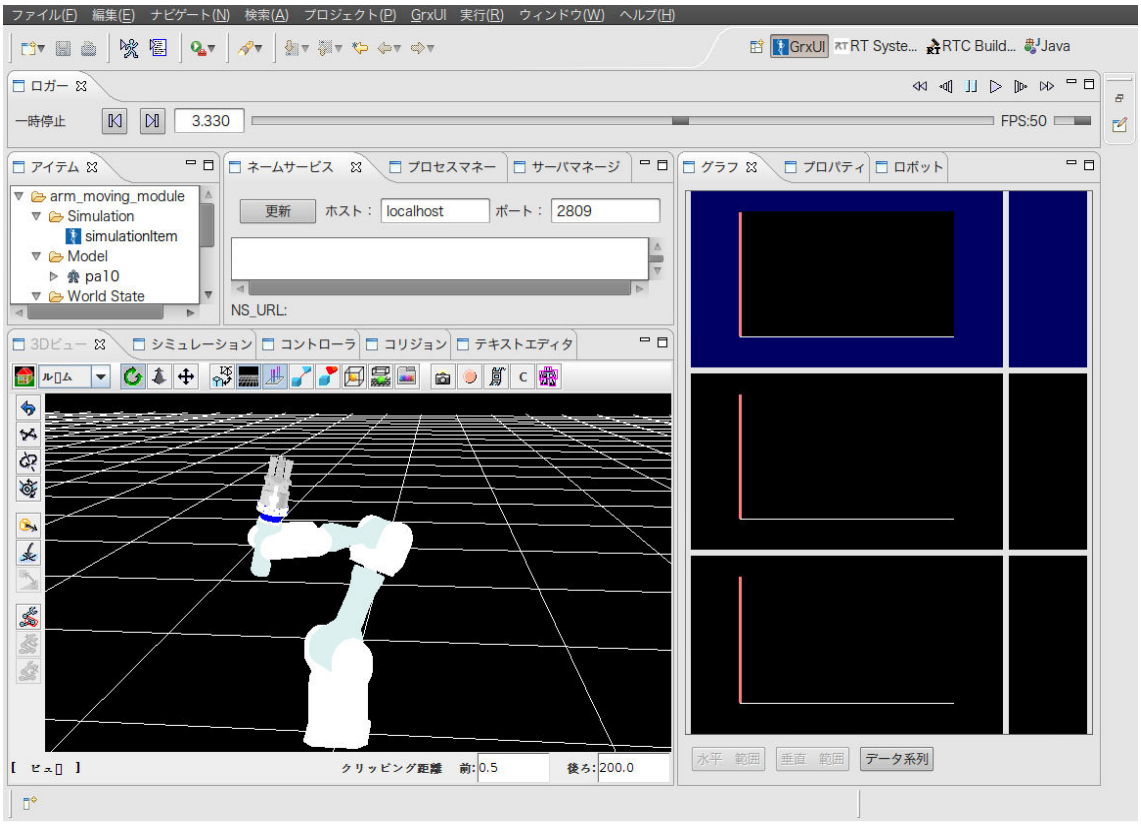

図 10 シミュレーションの実行中の表示

各モジュールを終了する場合は、redundancy_solution_module-20????ディレクトリ内で、"make shutdown"を実行してください。 rtcshell を使用して ArmMovingModule, ReduncancyMinimizeNorm, DirectionScenario モジュールを順に終了します。

```
$ make shutdown
```