# RT ミドルウェアの学習を目的とした
# 安価で入手容易なロボット上での実行環境の構築

# ユーザマニュアル

第 1.3 版

最終更新日　2012 年 3 月 15 日

埼玉大学工学部機械工学科　高橋直希

# 目　　次

# １　はじめに

本書は SI2011，RT ミドルウェアコンテスト 2011 応募作品「RT ミドルウェアの学習を目的とした安価で入手容易なロボット上での実行環境の構築」で使用されている RT ミドルウェア学習環境の仕様や構築方法などを詳細に解説したユーザマニュアルです．教育課程にて使用するテキストではありませんのでご注意下さい．

本応募作品は，OpenRTM-aist 公式 Web サイトにて以下の URL で公開されております．
http://www.openrtm.org/openrtm/ja/project/rtm_learning

また，本書で指定する「ホームページ」は以下の URL です．この URL 先にて各種ファイルをダウンロードすることができます．
http://design.mech.saitama-u.ac.jp/OpenRTM/

2 章は RT ミドルウェア学習環境，使用する RTC 群，使用するロボットの概要が述べられています．

3 章は RT ミドルウェア学習環境を使用する前の準備について述べられています．また，3.11 章のロボットの製作と 3.12 章のロボットの動作確認については，RT ミドルウェア学習環境に必須ではございません．シミュレーション環境のみでも RT ミドルウェア学習環境は動作しますので，どうしてもロボットを作ることができない場合は，3.11 章や 3.12 章は割愛頂いても構いません．

4 章は実際に RT ミドルウェア学習環境を使用する方法について述べられています．

5 章は本書のまとめが述べられています．

6 章は RT ミドルウェア学習環境の構築にあたって参考とした URL が述べられています．

3 章と 4 章を見れば，他の章の説明を読まなくとも RT ミドルウェア学習環境を使用することはできます．「使えればいい」という方や，「すべて見る時間がない」という方は，どうぞ 3 章からお読み下さい．

また，使用方法を学ぶだけなら，【教育課程関連情報】よりダウンロードできる「教育課程実施可能な環境の構築.pdf」や「テキスト.pdf」を使用したほうが容易に学習を行うことができます．本書はあくまでも「詳細なマニュアル」という位置づけであるため，まずは教育課程を試して頂くことをお勧めします．

# ２　概要

## ２．１　RT ミドルウェアとは？

RT ミドルウェアとは，Robot Technology Middleware の略称で，ロボットシステムのソフトウェア部分を構築するためのソフトウェアプラットフォーム（ソフトウェアの土台となる部分．OS のようなもの）です．

RT ミドルウェア上では RTC（Robot Technology Component の略称）を使用することができます．RTC とはロボットに関わる機能のソフトウェアモジュールであり，センサからの情報取得用 RTC，モータ制御用 RTC，カメラ制御用 RTC，画像処理アルゴリズムの RTC などたくさんの種類の RTC があります．RT ミドルウェア上で RTC を利用することでモジュール単位での並行開発や再利用，交換や更新などが容易に可能となります．また，ロボットシステム開発の際には，既存の RTC をできる限り再利用し，必要な部分のみ RTC を作成することでロボットシステムを構築することが可能となります．

よって，RT ミドルウェア上で RTC を使用することで，ロボットシステムのソフトウェア部分を低コストかつ効率的に構築できるようになります．

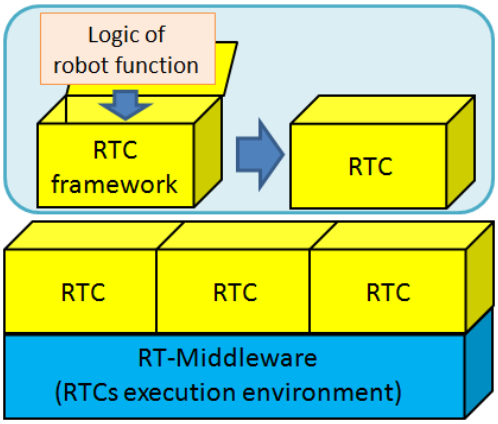


図 2.1.1　RT ミドルウェアと RTC のコンセプト

## ２．２　RTC とは？

RTC とは前述のとおり，ロボットに関わる機能のソフトウェアモジュールです．

ロボットに関わる機能は，RT ミドルウェアにおいて RT（Robot Technology の略称）機能要素と呼ばれており，あるまとまった機能を提供するロボット構成要素のことを指します．例えばサーボモータやセンサ，カメラといったデバイス単位や，こうしたデバイスの組み合わせにより実現される移動台車，アームなども RT 機能要素です．また，ハードウェアに直接的に結びつかなくとも，制御や画像処理のアルゴリズムなどソフトウェアのみで構成されるものも RT 機能要素として考えられています．

RTC は型，名称，データ長などが定められた入力ポート（InPort）と出力ポート（OutPort）を設定することができ，図 2.2.2 のように入力ポートと出力ポートを接続することで RTC

同士の通信を行います．他に「サービスポート」というポートもありますが，本学習環境で用いていないため解説を省略します．

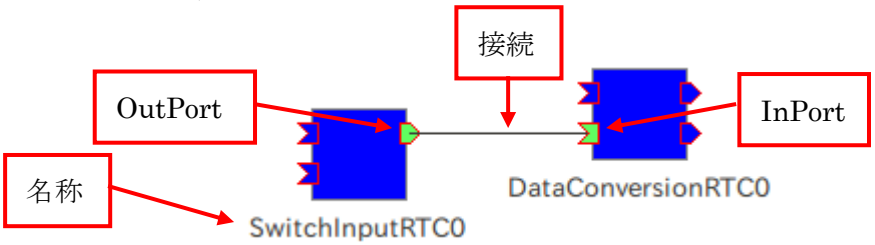


図 2.2.1　RTC の接続の説明

また，「コンフィグレーション」と呼ばれる，パラメータを動的に変更できる機能も有しています．一度出来上がった RTC でもコンフィグレーションによりパラメータ変更することで，自分の環境に応じた RTC をある程度作成することが可能となります．このことにより，RTC の再利用性や拡張性がより向上しています．

振る舞いを統一するための基本的な状態遷移である「アクティビティ」という機能もあり，これらの機能を利用することで，独立性や再利用性の高い RTC を容易に作成することができます．また，作成済みの RTC を再利用することで，最低限のプログラミングで，ロボットシステムのソフトウェア部分を構築することができます．

## ２．３　OpenRTM-aist とは？

OpenRTM-aist とは「Open-source and open architecture Robot Technology Middleware implemented by AIST」の略称であり（AIST とは National Institute of Advanced Industrial Science and Technology の略称であり，産業技術総合研究所のこと），産業技術総合研究所・知能システム研究部門によって開発された RT ミドルウェアの実装の一つです．

（　OpenRTM-aist の公式 Web サイト：http://www.openrtm.org/openrtm/ja/　）

特長として以下があげられます．

① オープンソースであり，無償で入手可能
② ネットワーク透過性，OS 非依存性，言語非依存性が重視されているため，分散オブジェクトミドルウェアである CORBA を用いて実装
③ RT System Editor や RTC Builder など各種ツール群により効率的に開発可能であり，GUI のツールも多く直感的に使用可能
④ 動力学シミュレータ OpenHRP3 を利用することが可能

RT ミドルウェアや RTC, OpenRTM-aist の説明に関しては, OpenRTM-aist の公式 Web サイト左のナビゲーション内にある「ドキュメント」にて解説されております．より詳しく知りたい方は，そちらをご覧下さい．

## ２．４　OpenHRP3 とは？

OpenHRP3 は，OpenHRP3 とは「Open-architecture Human-centered Robotics Platform version 3」の略称であり，ロボットのソフトウェア開発・シミュレーションのための統合ソフトウェアプラットフォームです．OpenHRP3 を用いることで，独自のロボットモデルと制御プログラムを動力学シミュレーションで検証することが可能となります．また，OpenRTM-aist に対応しており，ロボットモデルのコントローラを RTC として開発することが可能となっています．現在 OpenHRP3 公式 Web サイトにてオープンソースによる配布が行われており，無償で入手することができます．OpenHRP3 の利用には VRML ファイル（拡張子は wrl）を使用して表したシミュレーションモデルが必要であり，また，OpenRTM-aist も併せて使用する場合は RTC として開発したコントローラも用意する必要があります．

（　OpenHRP3 の公式 Web サイト：http://www.openrtp.jp/openhrp3/jp/index.html　）

OpenHRP3 を用いることで，シミュレーション内のロボットの関節角度や関節トルク，ワールド座標系における位置姿勢，距離データ，カメラ画像などを取得することができるようになります．また，それを出力し他の RTC に利用することができます．

OpenHRP3 は，
http://www.openrtp.jp/openhrp3/jp/install.html
にて動作確認が行われております．OS さえあれば，このページのインストール手順に従うことで，シミュレーション環境や RT ミドルウェア開発環境を構築することができます．

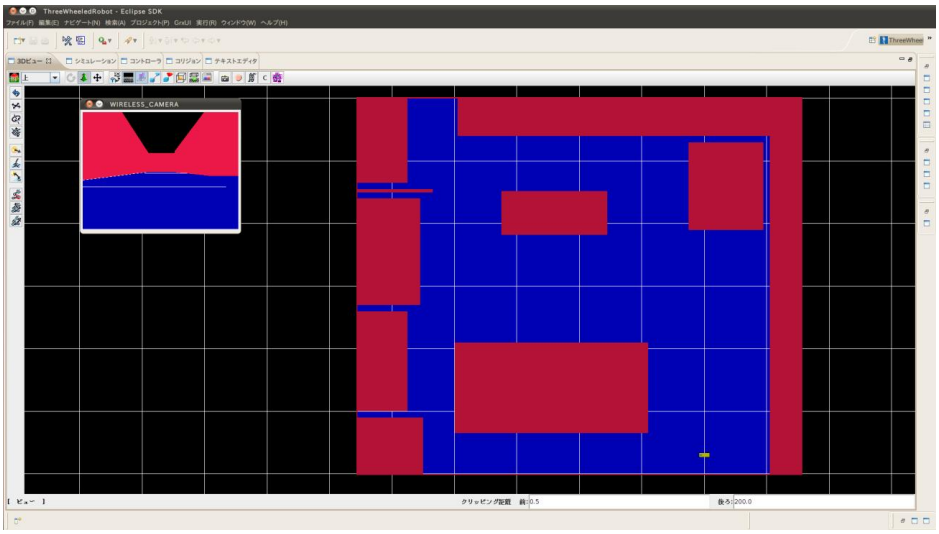


図 2.4.1　OpenHRP3 実行中の例
（左：ロボットモデル前方のカメラ画像，右：シミュレーション環境）

また，以下の URL にて OpenRTM-aist 公式 Web サイトでの公開も行われており，OpenHRP3 公式 Web サイトへのリンクが貼られています．
http://openrtm.org/openrtm/ja/project/openhrp3

## ２．５　リファレンスハードウェア・ソフトウェアの概要

RT ミドルウェアにおけるリファレンスハードウェアとは，前川製作所の山下らによって開発されたプラットフォームロボットを指します．リファレンスハードウェアは，電動移動台車の上に 7 自由度のマニピュレータ 1 台，内部に PC を搭載するといった基本構成となっています．また，これらのモジュール化を図り，それぞれ単独での動作も可能となっていることで，移動機構またはハンドリング機構が不要な研究用途にも対応できています．

リファレンスハードウェア上では，マニピュレータ部分による作業知能に関する RTC 群，電動移動台車部分による移動知能に関する RTC 群，コミュニケーション知能に関する RTC 群が動作可能となっています．これらの RTC 群の中から，次世代ロボット知能化技術開発プロジェクトで開発されたオープンソースの RTC を再利用し，システム構成例として提供した RTC 群が「来訪者受付システム」です．来訪者受付システムは，来訪者の入館・退館における受付業務を主としたオフィスサービスを行うものであり，各サービスのソフトウェア部分は RT ミドルウェアと RTC によって開発されています．OS は Ubuntu10.04LTS のみ対応しており，RT ミドルウェアは OpenRTM-aist-1.0.0-RELEASE と OpenRTM-aist-Python-1.0.0-RELEASE を，動力学シミュレータは OpenHRP Ver.3.1.1 を利用しています．また，来訪者受付システムは OpenRTM-aist 公式 Web サイトにてオープンソース形式で配布されており，無償でダウンロードすることが可能です．

（　来訪者受付システム公開 Web ページ：
http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001　）

また，ROBOSSA（Robot Software Suite, AIST の略称）と呼ばれるロボットシステム開発をサポートする産総研提供の統合ソフトウェアプラットフォームがあります．これは，RT ミドルウェア「OpenRTM-aist」，RTC 開発ツール群「OpenRTP-aist」，産総研が提供するオープンソースの知能ソフトウェアモジュール（RTC）群「OpenRTC-aist」から構成されるロボット開発用基盤ツールです．来訪者受付システム Ver1.0 は，OpenRTC-aist において移動知能モジュールパッケージのシステム構成例として採用されており，RT ミドルウェアにおいて移動知能モジュールを用いたオフィシャルのロボットソフトウェアシステムであると考えられます．

（　ロボット開発用基盤ツール「ROBOSSA」：http://robossa.org/　）
（　産総研が提供するオープンソース RTC 群「OpenRTC-aist」：http://openrtc.org/　）
（　来訪者受付システム Ver1.0 公開 Web ページ：
http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001_10　）

そこで本学習環境においては，OpenRTC-aist の移動知能モジュールパッケージのシステム構成例であり，リファレンスハードウェア上で動作する RTC 群「来訪者受付システム」をリファレンスソフトウェアだと想定します．

## ２．６　RT ミドルウェア学習環境の概要

今回開発した RT ミドルウェア学習環境は，RT ミドルウェアの導入教育を行うことを目的とした安価かつ入手容易で，多人数向けかつ初心者向けの，より実践的な RT ミドルウェア学習環境です．開発にあたり，前述の通りリファレンスハードウェア上で動作する RTC 群「来訪者受付システム Ver1.0」をリファレンスソフトウェアだと想定し，これを再利用することでリファレンスハードウェアの移動能力を再現しました．これにより，リファレンスハードウェアを使用した場合と同等の学習環境が実現でき，来訪者受付システムの再利用性の実証を行うことができました．

再利用した RTC 群は，来訪者受付システムの中でも，移動能力に関する RTC 群です．これは 5 の研究体により開発された 19 の RTC を指すものであり，ユーザによる目的地入力，経路計画・軌道追従，障害物検知・衝突回避，外界センサ（カメラ）による自己位置姿勢推定，走行系，オペレータ操作という分類の RTC 群で構成されているものです．しかし，来訪者受付システム Ver 1.0 では，正常に障害物検知・衝突回避を行うことができないという問題点があるため，本学習環境では障害物検知・衝突回避に関する RTC 群は再利用しないこととしました．

また，リファレンスソフトウェアの再利用によるリファレンスハードウェアの移動能力の再現のほか，RT ミドルウェア学習環境は以下を目標にして開発されました．

① 安価な移動ロボットを開発し使用すること
② 全情報を一般公開すること
③ RT ミドルウェアの導入教育を行うための教育課程を編成すること

そのために，まず，安価で製作容易な 3 輪移動ロボット（図 2.6.1）を開発しました．このロボットに関しては，2.9 章にて後述します．

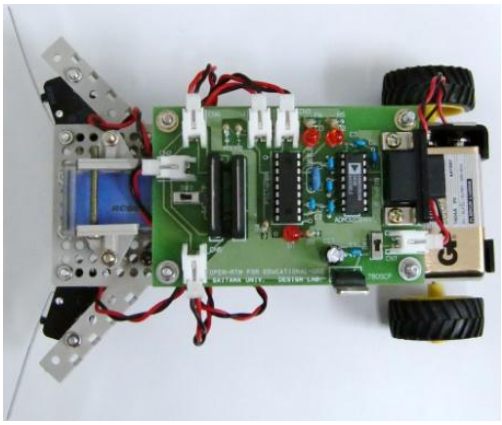

図 2.6.1　3 輪移動ロボット

また，全情報を以下の URL にて一般公開しました．公開した情報に関しては，以下の URL（「1 章　はじめに」で述べた URL と同一）をご覧下さい．一般公開することにより，本学習環境の入手・配布が容易となり，ハードウェア・ソフトウェア両面で自由な修正や

改善が可能となりました．なお，今後，ロボット教育関連情報を掲載する予定であるロボペディアでも公開を行う予定です．

（　OpenRTM-aist 公式 Web サイト「導入教育を目的とした安価で入手容易な RT ミドルウェア学習環境」公開 Web ページ：http://openrtm.org/openrtm/ja/project/rtm_learning　）

（　埼玉大学　設計工学研究室「導入教育を目的とした安価で入手容易な RT ミドルウェア学習環境」公開 Web ページ：http://design.mech.saitama-u.ac.jp/OpenRTM/　）

また，RT ミドルウェアの導入教育を行うための教育課程も編成しました．これは，上記 URL でダウンロードできる【教育課程関連情報】をご覧下さい．

以上の目標を達成し，開発した RT ミドルウェア学習環境のシステム構成を図 2.6.2 に示します．

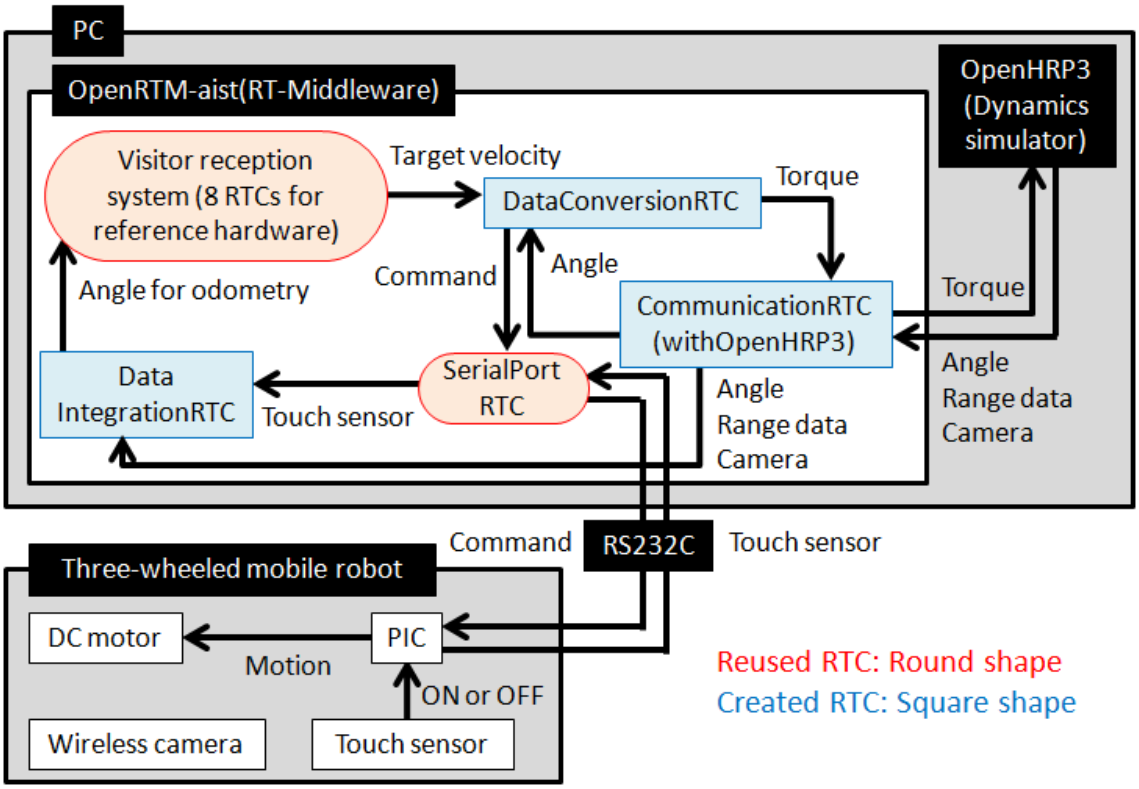


図 2.6.2　RT ミドルウェア学習環境のシステム構成

ハードウェアには 3 輪移動ロボット（2.9 章にて後述）を使用し，ソフトウェアには OpenRTM-aist と OpenHRP3 を使用しました．使用したバージョンは OpenRTM-aist-1.0.0-RELEASE と OpenHRP Ver.3.1.1 です．図 2.6.2 にて，再利用した RTC は，来訪者受付システムの移動能力に関する RTC 群「Visitor reception system (8 RTCs for reference hardware)」，3 輪移動ロボットとの通信用 RTC「SerialPortRTC」で

あり，新規に開発した RTC は，OpenHRP3 とのプロセス間通信用 RTC「CommunicationRTC(withOpenHRP3)」，来訪者受付システムと開発した RTC との接続用 RTC「DataConversionRTC」・「DataIntegrationRTC」です．OpenRTM-aist 上ではこれらの RTC を動作させます．これらの RTC 群の接続図を，図 2.6.3 に示します．

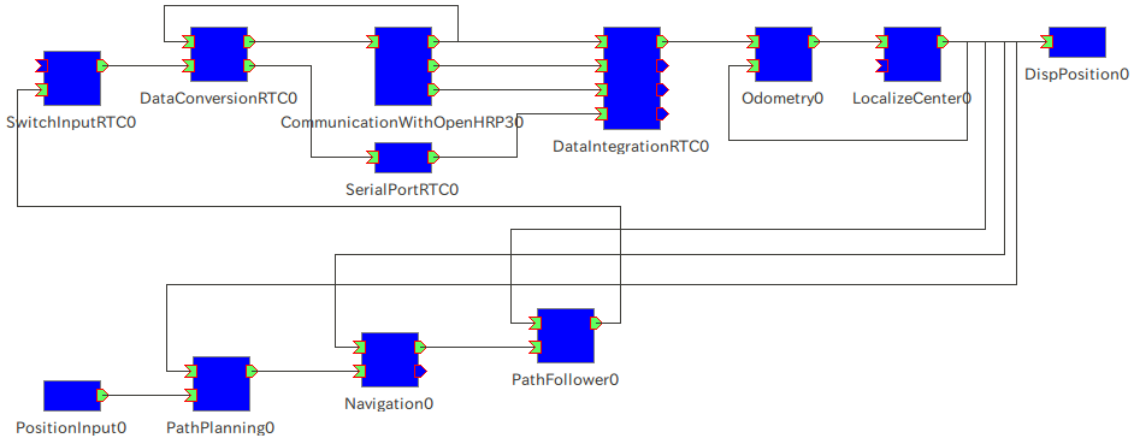


図 2.6.3　RT ミドルウェア学習環境の RTC 接続図

また，図 2.6.2 に示す通り，来訪者受付システムの再利用には，「Angle for odometry」，つまり，オドメトリのためのロボットの車輪回転角度が必要となります．しかし，本研究にて使用する 3 輪移動ロボットは，エンコーダを実装していないため，車輪の回転角度を計測することができません．

そこで，安価な 3 輪移動ロボットで取得できない情報は動力学シミュレータ OpenHRP3 を用いて生成しました．また，本システムでは用いていませんが，ロボットモデル前方に搭載したカメラのカラー画像や，測域センサによる距離データも生成することができます．これにより，安価なロボットでもリファレンスソフトウェアを利用可能となりました．

OpenHRP3 で使用するシミュレーションモデルは VRML ファイル（拡張子は wrl）であり，3 輪移動ロボットのシミュレーションモデルには図 2.6.4 を，床のシミュレーションモデルには図 2.6.5 を，障害物のシミュレーションモデルには図 2.6.6 の直方体モデルと図 2.6.7 の円柱モデルを使用します．これらを組み合わせることで，図 2.6.8 のようなシミュレーション環境を構築することが可能となります．

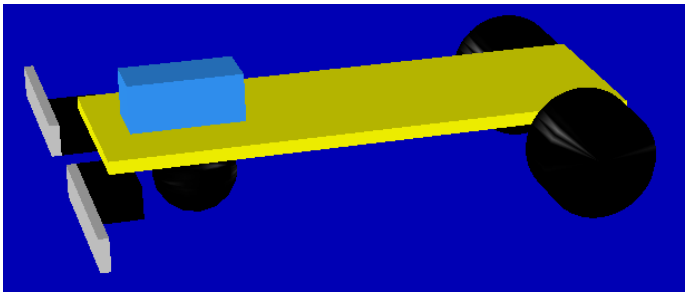

図 2.6.4　3 輪移動ロボットのシミュレーションモデル

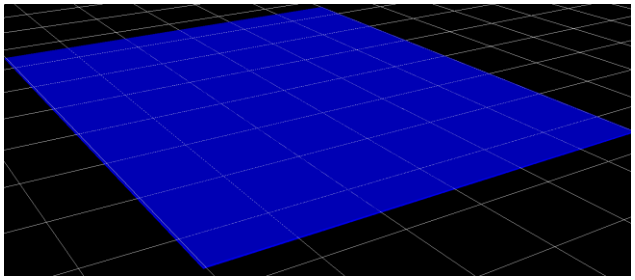

図 2.6.5　床のシミュレーションモデル

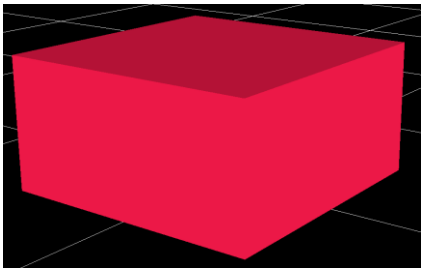

図 2.6.6　障害物に用いるための直方体モデル

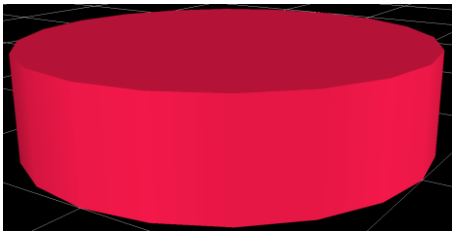

図 2.6.7　障害物に用いるための円柱モデル

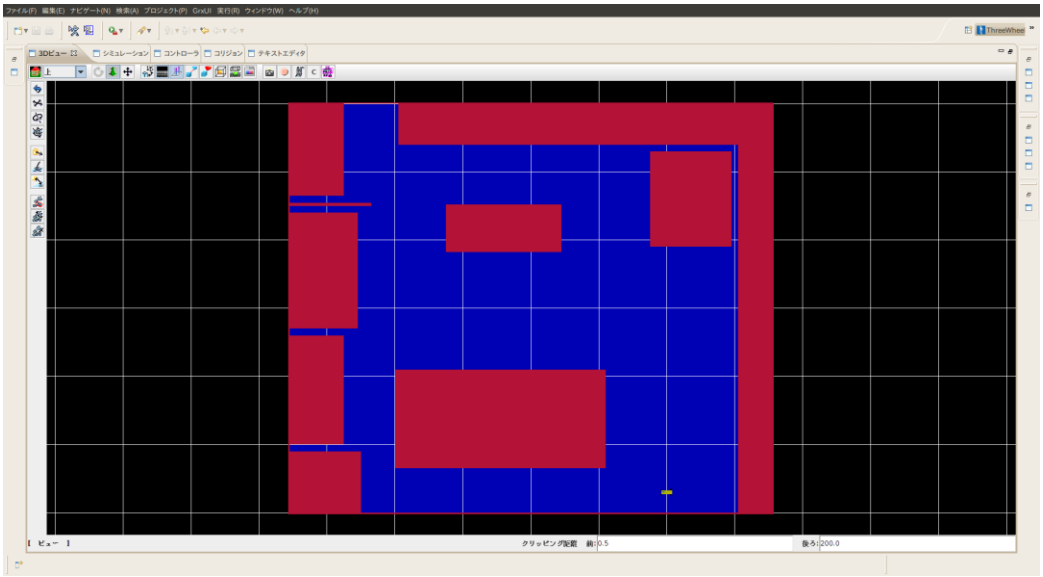

図 2.6.8　OpenHRP3 によるシミュレーション環境の例

リファレンスハードウェアと来訪者受付システムを用いた従来のシステム構成との違いは，動力学シミュレータ OpenHRP3 を使用しプロセス間通信用 RTC「CommunicationRTC(withOpenHRP3)」を開発したこと，安価なロボットとして 3 輪移動ロボットを使用しその通信用 RTC「SerialPortRTC」を開発したこと，来訪者受付システム・CommunicationRTC(withOpenHRP3)・SerialPortRTC とを接続するための RTC を開発したことです.

このシステムにより，3 輪移動ロボットは自律移動ができるようになりました.

## ２．７　再利用した RTC

### ２．７．１　来訪者受付システムの移動能力に関する RTC 群

来訪者受付システムの移動能力に関する RTC 群の内，本学習環境では自律移動に関する部分を再利用しています.

来訪者受付システムは,

http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001

にて公開されております．本学習環境では Ver 1.0 を用いているため,

http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001_10

のソース（RTC 群など）やドキュメント，参考資料などを使用しています．（RTC 群の実際の使用方法については 3.5 章以降で詳しく解説します）

上記 URL で公開されている来訪者受付システム内，本学習環境では移動能力に関する RTC 群を用います．その中で，本学習環境で使用している 8 個の RTC を紹介します．RTC の詳細は上記 URL にて公開されているドキュメント，参考資料を参照して下さい.

(1) PositionInput：目的地入力
(2) PathPlanning：経路計画
(3) Navigation：経路走行
(4) PathFollower：軌跡追従
(5) SwitchInputRTC：自律と操作の入れ替え（本学習環境では操作による移動は未実装）
(6) Odometry：オドメトリ推定
(7) LocalizeCenter：自己位置姿勢推定
(8) DispPosition：位置表示

また，RTC 群に入力される情報はロボットの左右後輪の回転角度であり，出力する情報は目標並進速度と目標旋回速度です．開発環境を表 2.7.1 に，群としてのデータポート（InPort）を表 2.7.2 に，群としてのデータポート（OutPort）を表 2.7.3 に示します.

表 2.7.1　来訪者受付システムの開発環境

| | |
|---|---|
| OS | Ubuntu 10.04 LTS |
| RT ミドルウェア | OpenRTM-aist-1.0.0-RELEASE |

表 2.7.2　来訪者受付システムのデータポート（InPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| CurrentWheelAngle | RTC::TimedDoubleSeq | 2 | ロボットの<br>左右後輪の<br>回転角度[rad] |

表 2.7.3　来訪者受付システムのデータポート（OutPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| outVel | IIS::TimedVelocity2D | 1 | 目標並進速度[m/s]と<br>目標旋回速度[rad/s] |

### ２．７．２　SerialPortRTC

SerialPortRTC は，RTC 群と 3 輪移動ロボットとで RS232C 通信を可能とする RTC です.

SerialPortRTC は，「ysuga.net」
http://ysuga.net/
にて公開されている「5.1COM ポートにアクセスする RTC 作成」
http://ysuga.net/robot/rtm/practice/com_port
を参考に開発しました.

本学習環境では，上記 URL の SerialPortRTC に多少の修正を加えており，修正内容は 3.4.1 章で詳しく解説します.

SerialPortRTC の開発環境を表 2.7.4 に, 動作条件を表 2.7.5 に, データポート（InPort）を表 2.7.6 に，データポート（OutPort）を表 2.7.7 に，RTC のコンフィグレーションを表 2.7.8 に示す.

表 2.7.4　SerialPortRTC の開発環境

| | |
|---|---|
| OS | Ubuntu 10.04 LTS |
| RT ミドルウェア | OpenRTM-aist-1.0.0-RELEASE |
| 開発言語 | C++ |
| コンパイラ | gcc 4.4.3 |

表 2.7.5　SerialPortRTC の動作条件

| | |
|---|---|
| 実行周期 | 100.0[Hz] |

表 2.7.6　SerialPortRTC のデータポート（InPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| in | RTC::TimedOctetSeq | 4 | 3 輪移動ロボットへの指令値<br>（RS232C 通信で送るデータ） |

表 2.7.7　SerialPortRTC のデータポート（OutPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| out | RTC::TimedOctetSeq | 12 | 3 輪移動ロボットから受け取った<br>タッチセンサの ON/OFF のデータ<br>（RS232C 通信で受け取るデータ） |

表 2.7.8　SerialPortRTC のコンフィグレーション

| 名称 | 型 | デフォルト値 | 説明 |
|---|---|---|---|
| debug | int | debug | 未使用 |
| filename | std::string | /dev/ttyUSB0 | 例：”/dev/ttyUSB0”，”COM0” |
| baudrate | int | 9600 | 例：9600，115200 |
| packet_size | int | 12 | 出力ポートから packet_size<br>ごとにデータを区切って送信 |

## ２．８　新規に開発した RTC

### ２．８．１　DataConversionRTC

DataConversionRTC は，入力された目標並進速度と目標旋回速度を，3 輪移動ロボットへ渡す動作指令値と，OpenHRP3 のロボットモデルへ渡すトルクに変換する RTC です.

DataConversionRTC は，来訪者受付システムの移動能力に関する RTC 群の中のひとつであるシミュレーション用の RTC「MotorControl」の，目標速度から OpenHRP3 のロボットモデルに与えるトルクを算出するアルゴリズムを参考に開発しています.

MotorControl とは，来訪者受付システム Ver 1.0
http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001_10
にある，Download の【ソース】の RH 関連（ファイル名 RH_10.zip）をダウンロードし，展開した中の，
RH/RemotePC/src/comp/ForSimulation/MotorControl
のことです.

また，移動ロボット用コンポーネント群である OpenINVENT（Ver.4.0.0 にて開発終了）にて MotorControl の説明がされている word ファイルなどがダウンロードできますので，参考にして下さい.

（　OpenINVENT の WEB ページ：http://www.openrtp.jp/INVENT/　）

開発環境を表 2.8.1 に，動作条件を表 2.8.2 に，データポート（InPort）を表 2.8.3 に，データポート（OutPort）を表 2.8.4 に，RTC のコンフィグレーションを表 2.8.5 に示す．

表 2.8.1　DataConversionRTC の開発環境

| | |
|---|---|
| OS | Ubuntu 10.04 LTS |
| RT ミドルウェア | OpenRTM-aist-1.0.0-RELEASE |
| 開発言語 | C++ |
| コンパイラ | gcc 4.4.3 |

表 2.8.2　DataConversionRTC の動作条件

| | |
|---|---|
| 実行周期 | 1000[Hz] |
| 備考 | CommunicationRTC(withOpenHRP3)<br>に RTC としての処理を委ねている<br>（詳細は 2.8.2 章にて説明） |

表 2.8.3　DataConversionRTC のデータポート（InPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| model_angle | RTC::TimedDoubleSeq | 2 | ロボットモデルの<br>左右後輪の回転角度[rad]<br>（タイムスタンプも使用） |
| target_velocity | IIS::TimedVelocity2D | 1 | 目標並進速度[m/s]と<br>目標旋回速度[rad/s] |

表 2.8.4　DataConversionRTC のデータポート（OutPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| model_torque | RTC::TimedDoubleSeq | 2 | ロボットモデルの左右後輪に<br>与えるトルク |
| robot_command | RTC::TimedOctetSeq | 4 | 3 輪移動ロボットへの指令値 |

表 2.8.5　DataConversionRTC のコンフィグレーション

| 名称 | 型 | デフォルト値 | 説明 |
|---|---|---|---|
| PGainL | double | 5.0 | 左車輪モータの P ゲイン |
| DGainL | double | 5.0 | 左車輪モータの D ゲイン |
| PGainR | double | 5.0 | 右車輪モータの P ゲイン |
| DGainR | double | 5.0 | 右車輪モータの D ゲイン |
| leftWheelID | int | 1 | 左車輪の ID 番号 |
| rightWheelID | int | 0 | 右車輪の ID 番号 |
| radiusOfLeftWheel | double | 0.018 | 左車輪の半径 |
| radiusOfRightWheel | double | 0.018 | 右車輪の半径 |
| lengthOfAxle | double | 0.078 | 左右車輪間の軸長さ |
| radiusOfBodyArea | double | 0.15 | ロボット全体を円で<br>囲んだ場合の半径<br>安全を考慮したエリア定義用 |
| torqueSensitivity | double | 1.0 | ロボットモデルへ与えるトルクの伝わりやすさ．1.0 推奨．大きいほど伝わりやすくなる． |
| torqueConstant | double | 1.0 | ロボットモデルへ与えるトルクを定数倍する値．1.0 推奨．大きいほどロボットモデルへ与えるトルクが大きくなる |

【注意点①】

DataConversionRTC の In Port にある IIS::TimedVelocity2D という型は来訪者受付システムにて使われている型で，intellirobot.idl 内で次のように定義されております．

```
struct TimedVelocity2D {
    RTC::Time tm;
    sequence<long> id;
    RTC::Velocity2D data;
    sequence<double> error;
};
```

DataConversionRTC では，

m_target_velocity.data.vx：目標並進速度[m/s]

m_target_velocity.data.vy：未使用

m_target_velocity.data.va：目標旋回速度[rad/s]

として扱っております．

詳しいIIS::TimedVelocity2Dの仕様は，3.5.1章でダウンロードするRH関連参考資料.zip内にある，移動SWG共通IF案（移動SWG共通IF案101008.pdf）をご覧下さい．

【注意点②】

コンフィグレーションのtorqueSensitivityとtorqueConstantは，再利用元であるMotorControlのアルゴリズムでは，このコンフィグレーションは存在しません．また，両方のコンフィグレーションとも1.0であることが望ましいです．もし実機とシミュレーションのロボットモデルの動きが一致しなければ，この値やモータのゲインを，実験を重ね細かい調整幅で調整して下さい．

【注意点③】

実機をより速く動作させたい場合は，DataConversionRTC.cpp内の275行目～321行目，つまり，「実機の動作設定　開始」から「実機の動作設定　終了」までを修正し再コンパイルして下さい．指令値に関しては，2.9.2章や3.12章で詳しく解説します．

### ２．８．２　CommunicationRTC(withOpenHRP3)

CommunicationRTC（withOpenHRP3）は，OpenHRP3のロボットモデルとプロセス間通信するためのRTCです．入力される情報はOpenHRP3のロボットモデルの左右後輪に与えるトルクです．出力する情報は，ロボットモデルの車輪回転角度，シミュレーション上の測域センサによる距離データ，ロボットの前方のカメラ画像です．

CommunicationRTC（withOpenHRP3）は，DataConversionRTCのディレクトリ内に置くスクリプトファイルを起動することで，DataConversionRTCとともに実行されます．このスクリプトにより，DataConversionRTCとしての処理をOpenHRP3側のControllerBridgeに委ねることができます．　DataConversionRTCなどCommunicationRTC（withOpenHRP3）と接続しているすべてのRTCが，OpenHRP3でのシミュレーションのスタート・ストップに連動して実行されるようになります．実際にスクリプトを作成する方法については，3.4.3章で詳しく解説します．

スクリプトの詳細については，OpenHRP3の公式Webサイトにある解説ページを参考にして下さい．

（　スクリプト解説ページ：http://www.openrtp.jp/openhrp3/jp/controller_bridge.html　）

CommunicationRTC（withOpenHRP3）の開発環境を表2.8.6に，動作条件を表2.8.7に，データポート（InPort）を表2.8.8に，データポート（OutPort）を表2.8.9に，RTCのコンフィグレーションを表2.8.10に示す．

表 2.8.6　CommunicationRTC（withOpenHRP3）の開発環境

| | |
|---|---|
| OS | Ubuntu 10.04 LTS |
| RT ミドルウェア | OpenRTM-aist-1.0.0-RELEASE |

表 2.8.7　CommunicationRTC（withOpenHRP3）の動作条件

| | |
|---|---|
| 実行周期 | 1000[Hz] |
| 実行コンテキスト | SynchExtTriggerEC |
| 備考 | OpenHRP3 の ControllerBridge に処理を委ねている |

表 2.8.8　CommunicationRTC（withOpenHRP3）のデータポート（InPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| torque | RTC::TimedDoubleSeq | 2 | ロボットモデルの左右後輪に与えるトルク |

表 2.8.9　CommunicationRTC（withOpenHRP3）のデータポート（OutPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| angle | RTC::TimedDoubleSeq | 2 | ロボットモデルの左右後輪の回転角度[rad] |
| image | RTC::TimedLongSeq | ピクセル数 | ロボットモデル前方のカメラのカラー画像 |
| range_sensor | RTC::TimedDoubleSeq | センサ出力数 | 測域センサから得られた距離データ[m] |

表 2.8.10　CommunicationRTC（withOpenHRP3）のコンフィグレーション

| 名称 | 型 | デフォルト値 | 説明 |
|---|---|---|---|
| — | — | — | コンフィグレーションなし |

### ２．８．３　DataIntegrationRTC

DataIntegrationRTC は，CommunicationRTC（withOpenHRP3）と SerialPortRTC の出力を統合して，データを来訪者受付システムの移動能力に関する RTC 群へ渡す RTC です．

入力される情報は，CommunicationRTC（withOpenHRP3）から出力された車輪回転角度，前方カメラのカラー画像，シミュレーション上の測域センサによる距離データと，SerialPortRTC から出力されたタッチセンサの ON/OFF のデータです．出力する情報は，データの意味（角度，カラー画像，距離データ，タッチセンサ ON/OFF）を変えずに，それらのデータに単位変換などの多少の修正を施したものとなっております．

DataIntegrationRTC の開発環境を表 2.8.11 に，動作条件を表 2.8.12 に，データポート（InPort）を表 2.8.13 に，データポート（OutPort）を表 2.8.14 に，RTC のコンフィグレーションを表 2.8.15 に示す.

表 2.8.11　DataIntegrationRTC の開発環境

| | |
|---|---|
| OS | Ubuntu 10.04 LTS |
| RT ミドルウェア | OpenRTM-aist-1.0.0-RELEASE |
| 開発言語 | C++ |
| コンパイラ | gcc 4.4.3 |

表 2.8.12　DataIntegrationRTC の動作条件

| | |
|---|---|
| 実行周期 | 1000000[Hz] |

表 2.8.13　DataIntegrationRTC のデータポート（InPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| in_angle | RTC::TimedDoubleSeq | 2 | ロボットモデルの<br>左右後輪の回転角度[rad] |
| in_color_image | RTC::TimedLongSeq | ピクセル数 | ロボットモデル前方の<br>カメラのカラー画像 |
| in_range_sensor | RTC::TimedDoubleSeq | センサ出力数 | 距離センサから得られた<br>距離データ[m] |
| in_touch_sensor | RTC::TimedOctetSeq | 12 | タッチセンサの<br>ON/OFF のデータ |

表 2.8.14　DataIntegrationRTC のデータポート（OutPort）

| 名称 | 型 | データ長 | 説明 |
|---|---|---|---|
| out_angle | RTC::TimedDoubleSeq | 2 | ロボットモデルの<br>左右後輪の回転角度[rad] |
| out_color_image | RTC::TimedLongSeq | ピクセル数 | ロボットモデル前方の<br>カメラのカラー画像 |
| out_range_sensor | RTC::TimedDoubleSeq | センサ出力数 | 測域センサから得られた<br>距離データ[mm] |
| out_touch_sensor | RTC::TimedState | 1 | タッチセンサの<br>ON/OFF のデータ |

表 2.8.15　DataIntegrationRTC のコンフィグレーション

| 名称 | 型 | デフォルト値 | 説明 |
|---|---|---|---|
| — | — | — | コンフィグレーションなし |

## ２．９　３輪移動ロボット

### ２．９．１　どんなロボットか

RT ミドルウェアを利用でき，安価であるため 20 人から 30 人程度の多人数向けの RT ミドルウェア学習にも使え，移動に関する最低限の機能を実装しているロボットとして，3 輪移動ロボット（3988 円）を開発しました．

3 輪移動ロボットは図 2.9.1 に示すロボットであり，画像縦が 105mm，画像横が 185mm，高さが 90mm という大きさのロボットとなっています．

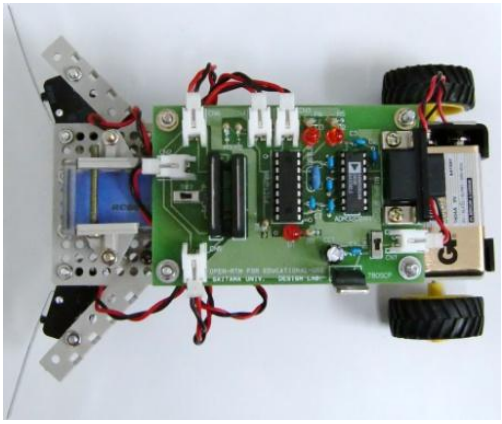

図 2.9.1　3 輪移動ロボット

選定した主な部品などを表 2.9.1 に示します．

表 2.9.1　選定した主な部品など

| | |
|---|---|
| マイコン | PIC16F88 |
| センサ | タッチセンサ用マイクロスイッチ<br>（オプション：ワイヤレスカメラ） |
| アクチュエータ | DC モータ |
| 外部との通信方法 | RS232C |

マイコンとして PIC16F88 を用いた理由は，PIC は安価であるためと開発環境を無料で利用できるためです．また，開発のための資料が豊富であり，独自に機能を追加・拡張することが容易であるためでもあります．

センサとしては，タッチセンサ用マイクロスイッチを使用します．これにより，マイクロスイッチの ON/OFF で容易に接触検知が行えるようになります．また，オプションとして，ワイヤレスカメラを実装できるようにしています．

アクチュエータとして DC モータを使用します．工作部品として広く使用されており，安価かつ入手容易で PIC による制御もモータドライバを介することで容易となるためです．

外部との通信に RS232C 通信を利用する理由は，使用する PIC16F88 は RS232C 通信を使用できる機能が備わっており，RS232C 通信ケーブルを用いることで容易にデータの送受信が行えるためです．

また，オプションとして無線カメラを搭載できるスペースをロボット前方に用意しました．

3輪移動ロボットのオプションである無線カメラなどを除いた総額は 3,988 円と安価です．3 輪移動ロボットの簡略化した部品表は表 2.9.2 に示します．また，3 輪移動ロボットのハードウェア情報（詳細な部品表，電子回路図，プリント基板の配線図，PIC プログラム）はホームページにてダウンロードすることができます．なお，RS232C インタフェース IC を購入した際，セラミックコンデンサと IC ソケット 16PIN は無料で付属したため，それを考慮して総額を計算しました．

表 2.9.2　3 輪移動ロボットの部品表とコスト

| 部品の種類 | 品番等 | 個数 | 価格 |
|---|---|---|---|
| PICマイコン | PIC16F88-I/P | 1 | 200 |
| モータドライバ | TA7291P | 2 | 200 |
| 3端子レギュレータ(5V1A) | L7805CV | 1 | 25 |
| 電解コンデンサ(電源用,100μF) | 35ZLH100M | 1 | 20 |
| セラミックコンデンサ(0.1μF) | RPEF11H104Z2K1A01B | 9 | 40 |
| セラロック(10MHz) | CSTLS10M0G53-B0 | 1 | 20 |
| カーボン抵抗(炭素皮膜抵抗)1/4W1kΩ | RD25S 1K | 6 | 6 |
| 赤色LED(Φ5) | OSDR5113A | 3 | 12 |
| ICソケット16PIN(ADM3202AN用) | 2227シリーズ | 1 | 0 |
| ICソケット18PIN(PIC用) | 2227シリーズ | 1 | 10 |
| ICソケット20PIN(モータドライバ用) | 2227シリーズ | 1 | 10 |
| スライドスイッチ(電源用) | SS12D01G4 | 2 | 50 |
| 電池ボックス(単三×4本,リード線付) | P-02671 | 1 | 70 |
| 電池ボックス(角型,リード線付) | BH-9V-1A | 1 | 60 |
| 単三型1.5V電池 | GLR6A | 4 | 80 |
| 角型9V電池 | GL6F22A | 1 | 100 |
| D-Sub コネクタ(9ピン,メス) | RDED-9S-LNA | 1 | 50 |
| RS232Cインタフェース IC | ADM3202AN | 1 | 150 |
| 横コネクタ オス(2ピン) | HRS DF1BZ-2P-2.5DS | 6 | 114 |
| 横コネクタ メス(2ピン) | HRS DF1B-2S-2.5R | 6 | 60 |
| ソケット用圧着端子 | HRS DF1B-2428SC | 12 | 47 |
| タッチセンサ用 マイクロスイッチ | V-103-1A4 | 2 | 610 |
| プリント基板 | 2層プリント基板 | 1 | 794 |
| ギアボックス | ツインモーターギヤーボックス | 1 | 700 |
| タイヤ | トラックタイヤセット | 1 | 150 |
| ボールキャスタ | ボールキャスター | 1 | 150 |
| 板 | ユニバーサルプレート | 1 | 260 |
| | | 総額: | 3988 |

また，006P 電池ボックスを電池スナップに，タッチセンサ用マイクロスイッチをレバーの長さが短いものに，ユニバーサルプレートは安価な木材に変更することで，総額を 3300 円程度とすることもできるように，より安価になるよう部品を選定し直すことが可能です．また，部品には入手容易な市販品を使用していることからも，3 輪移動ロボットには多人数向け教育に使用可能であるという利点があります．

ハードウェアを導入する理由は，ハードウェア特有の誤差を学習者に体感してもらうためと，ソフトウェアのみの場合に比べロボットに対する現実感を持たせることができ，より学習効果が高まると予想できるためです．

なお，3 輪移動ロボットを製作するために必要な情報は 3.11 章にて詳しく解説します．

### ２．９．２　ロボットが受け取れる情報

PWM により左右モータを制御する際のデューティ比を，55 通りに分類した指令値の内 1 つを受け取ることができます．その指令値を PIC プログラム内での定義に従って変換することで左右モータの動作決定を行うことができます．

受け取れる指令値の具体的な仕様について解説します．指令値は，「速度を決定する文字データ」＋「動作を決定する文字データ」＋「o」＋「k」という文字配列データとなるよう PIC プログラム内で定義されています．これにより，左右の DC モータの回転方向および 5 段階の速度の制御を行うことができるようになります．表 2.9.3 に速度を決定する文字データを，表 2.9.4 に動作を決定する文字データを示します．

表 2.9.3　速度を決定する文字データ

| 速度 | 受け取るデータ |
|---|---|
| 最大速度 | 1 |
| 高速 | 2 |
| 通常速度 | 3 |
| 低速 | 4 |
| 最低速度 | 5 |

表 2.9.4　動作を決定する文字データ

| 動作 | 受け取るデータ |
|---|---|
| 前進 | a |
| 後退 | b |
| 右折 | c |
| 急な右折 | d |
| 左折 | e |
| 急な左折 | f |
| 右後退 | g |
| 急な右後退 | h |
| 左後退 | i |
| 急な左後退 | g |
| 停止 | k |

表 2.9.3 と表 2.9.4 に示す文字データと「o」と「k」, つまり最大速度の前進を表す「1aok」, 最低速度の後退を表す「5bok」, 停止を表す「1kok」などを文字配列データとして受け取ることができます.

### ２．９．３　ロボットが送り出せる情報

ロボット前面に取り付けたタッチセンサ用マイクロスイッチの ON/OFF, つまりタッチセンサの接触の有無を送り出すことができます.

送り出せる情報を表 2.9.5 に示します.

表 2.9.5　送り出せるタッチセンサ情報

| タッチセンサの状態 | 送り出すデータ |
|---|---|
| ON | 1 |
| OFF | 0 |

送り出すことができる情報であるタッチセンサの反応の有無は, 表 2.9.5 に示す文字データとなって外部に送信されるよう PIC プログラム内で定義されています.

# ３　RT ミドルウェア学習環境を使う前の準備

本章では，以下のようなマークを使用しています．

（A）：新規に作成するもの

（B）：再利用するために行うべき作業

（C）：ハードウェア環境やソフトウェア環境，RT ミドルウェア実行環境などを種々の環境を変えるたびに必要な作業

（D）：不備があるために修正する必要がある作業

これらのマークを参考にした上で，本章をお読み下さい．

## ３．１　走行フィールドの確認（C）

走行フィールドに関する以下の事項の確認を行って下さい．

・3 輪移動ロボットが走行できる幅や距離は十分にとれているか

・地面の凹凸は極力少ないか

・地面の摩擦は十分であるか

・PC から半径 5m以内に走行フィールドが収まっているか（USB の規格上，USB2.0 延長ケーブルの最大長が 5m のため．USB ハブを介す，またはリピータ機能付きの USB 延長ケーブルを使用するのであれば，それ以上の長さが可能）

## ３．２　PC のスペック・OS の確認

まず，PC のスペックの確認を行います．動力学シミュレータ OpenHRP3 がなめらかに動くスペックの PC を用意して下さい．OpenHRP3 の推奨スペックなどは公開されておりませんが，OpenHRP3 のユーザーズフォーラムにて PC のスペックに関する質問がなされております．

OpenHRP3 質問用掲示板の案内

http://www.openrtp.jp/openhrp3/jp/forum.html

にて，OpenHRP3 ユーザーズフォーラムにログインします．そして，右上の「検索」より，検索モードは「AND」，検索記事は「現記事」の「全文」，表示モードは「本文なし」で，検索文字列を「PC のスペック」として検索します．すると PC のスペックに関する質問が表示されます．最新の情報ではありませんが，各項目をご覧頂くことで，PC のスペックの参考になると思われます．

また，OS の確認も行います．本学習環境は，Ubuntu Linux 10.04 LTS でのみ動作確認を行っているため，他の OS や Ubuntu でも他のバージョンを使用している方は，以下のページで Ubuntu Linux 10.04 LTS の導入を行って下さい．

Ubuntu Japanese Local Community Team の Web サイト

http://www.ubuntulinux.jp/

の左にある「Ubuntu の入手」から，「日本語 RemixCD イメージのダウンロード」を選択し，「Ubuntu 10.04.1 LTS - 2013 年 4 月までサポート」で CD イメージか Torrent ファイルより導入を行って下さい．

### ３．３　OpenHRP3 のインストール

OpenHRP3 をインストールします．本環境では Ver.3.1.1 をインストールします．

OpenHRP3 の公式 Web サイトである

http://www.openrtp.jp/openhrp3/jp/

にはインストールの手順が掲載されていますが，OpenHRP Ver.3.1.1 の対応バージョンである OpenRTM-aist-1.0.0 ではなく，OpenRTM-aist-1.1.0-rc3 がインストールされてしまいます．よって，以下に OpenRTM-aist-1.0.0 をインストールできる手順を示します．

まず，

http://www.openrtp.jp/openhrp3/jp/

へ行き，左の「OpenHRP3 他バージョン」を選択します．その後，「3.1 系ドキュメンテーション」の「バージョン 3.1.1」より，「3.1.1 ドキュメンテーショントップ」のリンクを選択して下さい．すると，

http://www.openrtp.jp/openhrp3/3.1.1_3.0.8/jp/index.html

に移動します．

そして，左の「インストール（Ver.3.1.1）」を選択します．次に，ページ中央の「インストール手順」から「Ubuntu Linux におけるインストール」にある「バイナリパッケージからインストール」を選択します．そして，「1. インストール前確認」を行った後，ページ中央の「apt-get の使用」の通り，「/etc/apt/source.list」に

deb http://www.openrtm.org/pub/Linux/ubuntu/ lucid main

と，

deb http://archive.canonical.com/ubuntu lucid partner

の行を追加します．これは，Ubuntu のメニューから「システム＞システム管理＞ソフトウェア・ソース」にて，「他のソフトウェア」タブ内で「＋追加」を選択し，上記 2 行を追加しても同様の結果となります．

次に，以下のコマンドを実行し OpenRTM-aist-1.0.0 と OpenHRP3 のインストールを行います．

```
$sudo apt-get update
$sudo apt-get install openrtm-aist=1.0.0-2
$sudo apt-get install openrtm-aist-dev=1.0.0-2
$sudo apt-get install openrtm-aist-doc=1.0.0-2
$sudo apt-get install openrtm-aist-example=1.0.0-2
$sudo apt-get install openhrp3.1=3.1.1-
```

これでインストールが行われました．インストールが行われたことを確認するために，以下のコマンドを入力して下さい．

$dpkg -l ‘openrtm*’

$dpkg -l ‘openhrp*’

これにより，openrtm-aist, openrtm-aist-dev, openrtm-aist-doc, openrtm-aist-example はバージョンが 1.0.0-2 に，openhrp3.1 はバージョンが 3.1.1-になっていることが確認でき，インストールが行われたことが確認できます．

なお，OpenRTM-aist をバージョン 1.0.0 のままで固定し OpenRTM-aist のアップデートを今後行わないようにしたい場合は，先ほどの「/etc/apt/source.list」や「システム＞システム管理＞ソフトウェア・ソース」を開き，「他のソフトウェア」タブ内で

deb http://www.openrtm.org/pub/Linux/ubuntu/ lucid main

のチェックを外し，「閉じる」を行って下さい．apt-get で OpenRTM-aist や OpenHRP3 のアップデートやアップグレードされなくなります．

次に，OpenHRP3 公式 Web サイトにて先ほどの「バイナリパッケージからインストール」から，「Java のサードパーティ確認」と「Eclipse、および、GrxUI プラグインのインストール」を行います．この際，Eclipse の解凍先は「/home/(アカウント名)」内にして下さい．そして，最下部の「GrxUI の起動・初期設定」に進みます．

「GrxUI の起動・初期設定」においては，全ての項目を指示通りに行って下さい．指示通り行う際，

Ubuntu9.04 以降の環境でサンプルが動作しない（Linux 対象）：

http://www.openrtp.jp/openhrp3/jp/troubleshooting.html#error_ipv6_localhost

と，Ubuntu9.10 以降の環境での Eclipse の起動：

http://www.openrtp.jp/openhrp3/jp/init_grxui.html#ubuntu910

を忘れないようにして下さい．また，「Ubuntu9.10 以降の環境での Eclipse の起動」においては，xulrunner のバージョンを適宜設定して下さい．なおこのとき，Eclipse の起動スクリプトの名称は，「script.sh」として下さい．また，Eclipse を最初に起動する際，まず，workspace の場所を設定することになります．これは，デフォルトで指定されている通り「/home/(アカウント名)」内に設定して下さい．

インストールが完了したら，「GrxUI の起動・初期設定」の下にある「サンプルシミュレーションの実行」により無事インストールできているかを確認して下さい．無事インストールできていたら，OpenHRP3 のインストールは終了です．

OpenHRP3 のインストールにより，RT ミドルウェア「OpenRTM-aist Ver.1.0.0」，統合開発環境「Eclipse」，OpenHRP3 を GUI で操作できる「GrxUI」などを導入することができます．

## ３．４　RTC の開発，ダウンロードなど

### ３．４．１　SerialPortRTC

SerialPortRTC は，「ysuga.net」

http://ysuga.net/

にて公開されている「5.1COM ポートにアクセスする RTC 作成」

http://ysuga.net/robot/rtm/practice/com_port

を参考に開発します．

3.3 章の「Ubuntu9.10 以降の環境での Eclipse の起動」で作成したシェルスクリプト（script.sh）を実行し，Eclipse を起動します．

まず，「アプリケーション＞アクセサリ＞端末」で，端末を起動します．（図 3.4.1）

端末は，今後何度も起動するため，起動方法を覚えておいて下さい．

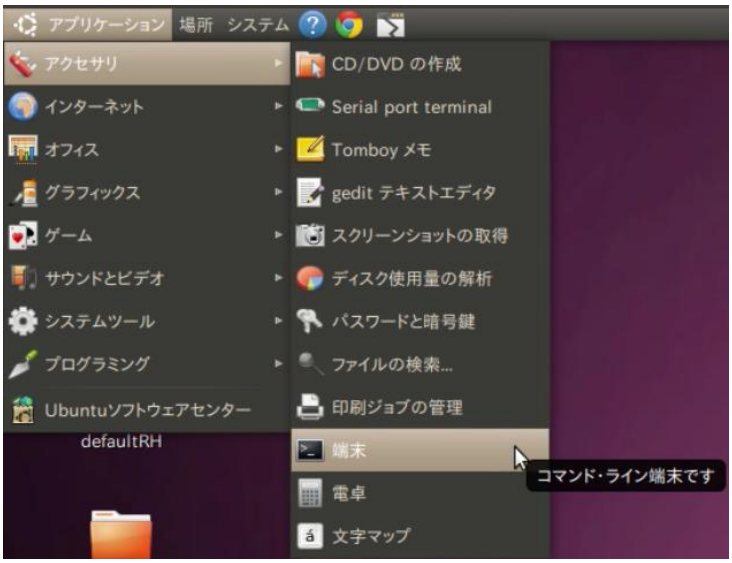


図 3.4.1　端末の起動

次に，Eclipse を実行します．

$cd /home/(アカウント名)/eclipse

$sudo sh script.sh

なお，稀に Eclipse は強制終了することがあります．その際は，上記 2 行のコマンドで再度，Eclipse を起動させて下さい．

次に，Eclipse のメニューにて図 3.4.2 のように，「ウインドウ＞パースペクティブを開く＞その他」を選択します．

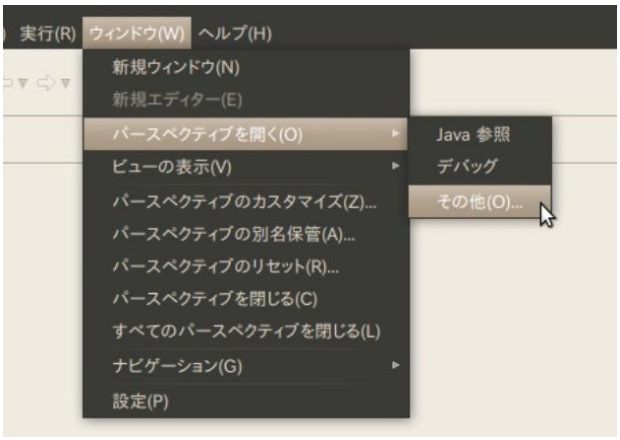


図 3.4.2　パースペクティブを開く

そして，その中から図 3.4.3 のように「RTC Builder」を選択します.

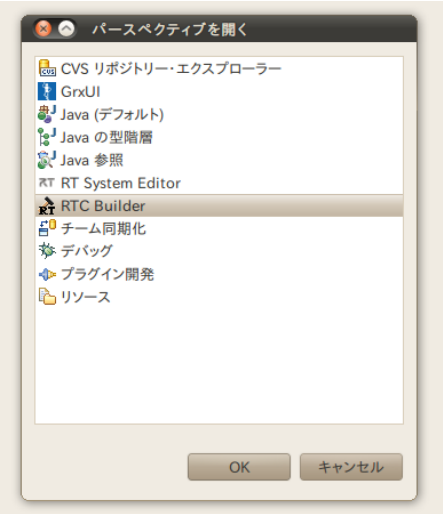


図 3.4.3　RTC Builder の選択

これで, RTC Builder が開かれます. RTC Builder を起動したら, 図 3.4.4 のように「Open New RtcBuilder Editor」を選択し，「ysuga.net」の「5.1COM ポートにアクセスする RTC 作成」
http://ysuga.net/robot/rtm/practice/com_port
の「1.RTC Builder によるコードの生成」通りに項目を入力していきます（図 3.4.5～図 3.4.9）.

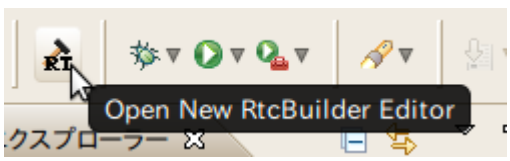


図 3.4.4　「Open New RtcBuilder Editor」の選択


図 3.4.5　基本


図 3.4.6　アクティビティ


図 3.4.7　データポート


図 3.4.8　コンフィグレーション


図 3.4.9　言語・環境

次に，コンフィグレーションの項目の修正を行います．表 3.4.1（表 2.7.8 と同一）のように packet_size のコンフィグレーションを新たに追加し，filename と baudrate のデフォルト値を変更して下さい．

表 3.4.1　SerialPortRTC のコンフィグレーション

| 名称 | 型 | デフォルト値 | 説明 |
|---|---|---|---|
| debug | int | debug | 未使用 |
| filename | std::string | /dev/ttyUSB0 | 例：”/dev/ttyUSB0”，”COM0” |
| baudrate | int | 9600 | 例：9600，115200 |
| packet_size | int | 12 | 出力ポートから packet_size ごとにデータを区切って送信 |

また，データポートで InPort として設定した「in」の表示位置を RIGHT から LEFT に変更して下さい．

これらの項目を入力し終えたら図 3.4.10 のように，「基本」タブにある「コード生成とパ

ッケージ化」により「コード生成」を行います.

すると，図 3.4.11 のように「指定されたプロジェクトが Workspace 内に存在しません。新規に生成してもよろしいでしょうか？」と聞かれるので「はい」とし，図 3.4.12 の「Generate success」という生成終了のウインドウの表示を待って下さい. 生成し終えたら，Eclipse を終了して構いません.

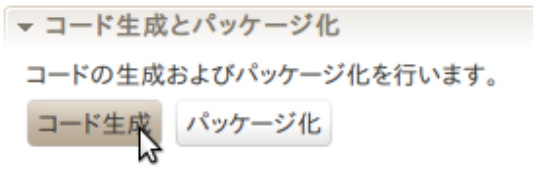


図 3.4.10　コード生成

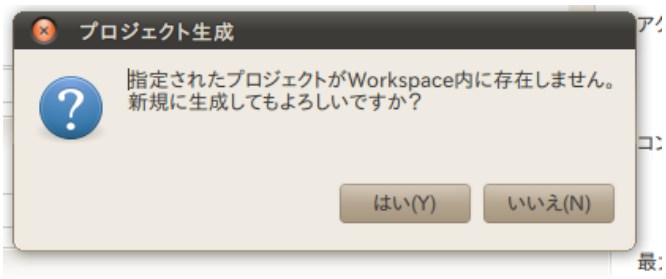


図 3.4.11　プロジェクトの新規生成

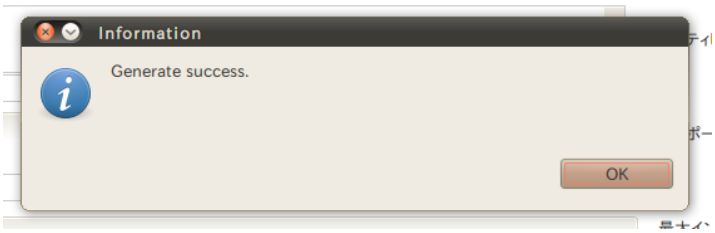


図 3.4.12　Generate success

次に,「2.コードの編集」です. 生成した SerialPortRTC のディレクトリ内に SerialPort.h と SerialPort.cpp を作成し，また，SerialPortRTC.h と SerialPortRTC.cpp の編集も行います.

「アプリケーション＞アクセサリ＞端末」で，端末を起動します.

次に，端末上で SerialPortRTC ディレクトリに移動します.

$cd /home/(アカウント名)/workspace/SerialPortRTC

そして,

$sudo gedit SerialPort.h SerialPort.cpp SerialPortRTC.h SerialPortRTC.cpp

と$以降を入力してファイルの作成，編集を行います．編集する内容は「ysuga.net」の「5.1COM ポートにアクセスする RTC 作成」にて示している通り入力して下さい．入力し

終わったら，図 3.4.13 に示す通り「保存」を選択し，ファイルを上書き保存して下さい．今後，gedit にてファイルの修正を行った後は，全て図 3.4.13 のように上書き保存を行うことを忘れないで下さい．

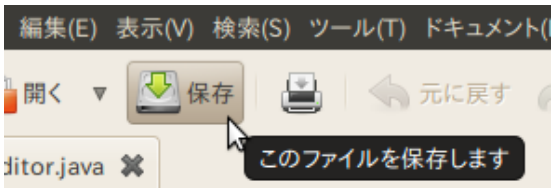


図 3.4.13　修正したファイルの保存

また，修正が終了し保存したら，図 3.4.14 に示す通り「×」をクリックし，テキストエディタを閉じて下さい．また，これも今後，gedit にてファイルの修正を行った後は，全て図 3.4.14 のようにテキストエディタの終了を行うことを忘れないで下さい．


図 3.4.14　テキストエディタの終了

次に，SerialPortRTC.h と Makefile.SerialPortRTC の修正を行います．この内容は，「ysuga.net」の「5.1COM ポートにアクセスする RTC 作成」に記載されていない編集内容です．

まず，上と同じように SerialPortRTC ディレクトリ内に端末上で移動し，SerialPortRTC.h と Makefile.SerialPortRTC をテキストエディタにて開きます．

$cd /home/(アカウント名)/workspace/SerialPortRTC

$sudo gedit SerialPortRTC.h Makefile.SerialPortRTC

SerialPortRTC.h の編集

「ysuga.net」の「5.1COM ポートにアクセスする RTC 作成」にて指定した場所を修正します．（D）

net::ysuga::CSerialPort *m_pSerialPort; // この行を追加

から，「C」を削除し，

net::ysuga::SerialPort *m_pSerialPort; // この行を追加

として下さい．

Makefile.SerialPortRTC の編集
31 行目を編集（D）
OBJS　　= SerialPort.o SerialPortRTC.o $(SKEL_OBJ) $(STUB_OBJ) $(IMPL_OBJ)
64 行目（SerialPortRTCComp.o: ～～ の下）に追加（D）
SerialPort.o: SerialPort.cpp SerialPort.h

これで編集は終了です．保存し，テキストエディタを終了して下さい．
次にコンパイルを行います．
まず，上と同じように SerialPortRTC ディレクトリ内に端末上で移動し，コンパイルを行います．
$cd /home/(アカウント名)/workspace/SerialPortRTC
$sudo make -f Makefile.SerialPortRTC clean
$sudo make -f Makefile.SerialPortRTC
これでエラーが発生しなければ，SerialPortRTC の完成です．

### ３．４．２　DataConversionRTC

DataConversionRTC はホームページにてダウンロードできる「ユーザマニュアル関連情報.zip」内にあります．
（　ホームページ：http://design.mech.saitama-u.ac.jp/OpenRTM/　）
ダウンロードした後，
/usr/share/OpenHRP-3.1/sample/controller/
内にコピーして下さい．
端末を開き，
$sudo cp -p -r (DataConversionRTC の場所) /usr/share/OpenHRP-3.1/sample/controller/
と入力することでコピーできます．

### ３．４．３　CommunicationRTC(withOpenHRP3)

CommunicationRTC は，DataConversionRTC 内に bridge.sh としてスクリプトファイルがすでにダウンロードされていますので，別途ダウンロードの必要はありません．

### ３．４．４　DataIntegrationRTC

DataIntegrationRTC はホームページにてダウンロードできる「ユーザマニュアル関連情報.zip」内にあります．
ダウンロードした後，workspace 内にコピーして下さい．端末を開き，
$sudo cp -p -r (DataIntegrationRTC の場所) /home/(アカウント名)/workspace/
と入力することでコピーできます．

## ３．５　来訪者受付システムの導入

### ３．５．１　来訪者受付システムのダウンロード

来訪者受付システム Ver 1.0 をダウンロードします.

http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_SYS001_10

にて，Download の【ソース】にある，

内容：RH 関連，ファイル：RH_10.zip

をダウンロードして下さい.

適宜，以下の関連文書のダウンロードを行って下さい．これらはダウンロードの手順には必要ありません．来訪者受付システムを用いる際の参考資料となります.

【ドキュメント】

内容：統合検証 RTC 開発ガイドライン，ファイル：ガイドライン.pdf

内容：RH 動作機能仕様書，ファイル：RH 動作機能仕様書 10_10.pdf

内容：RH 詳細設計書，ファイル：RH 詳細設計書 10_10.pdf

内容：RH 取扱説明書，ファイル：RH 取扱説明書 10_05.pdf

【参考資料】

内容：RH 関連(全部)，ファイル：RH 関連参考資料.zip

また，RH_10.zip はホームページにてダウンロードできる「教育課程関連情報.zip」内の Preparation 内にある「RH」ディレクトリを使用しても構いません.

### ３．５．２　各種インストール

まず，OpenCV2.0 のインストールを行います.

```
$sudo apt-get install libcv4 libhighgui4 libcvaux4 libcv-dev libhighgui-dev libcvaux-dev python-opencv opencv-doc
```

次に，gnuplot のインストールも行います

```
$sudo apt-get install gnuplot
```

これで各種インストールは終了です．次は RTC 群のダウンロード，修正，コンパイルです.

### ３．５．３　移動能力に関する RTC 群の修正

以下の手順通りに RTC 群の修正などを行って下さい.

（１）zip ファイルの展開，コピー

3.5.1 章でダウンロードした RH_10.zip を適当な場所に展開します．展開すると「RH」というディレクトリが出てきますので，それを/opt/にコピーします.

端末を開き，

```
$sudo cp -p -r (コピー元の RH ディレクトリの場所) /opt/
```

と入力することでコピーすることができます．コピーすることで，/opt/RH というディレクトリが生成されます.

（２）RHBase ディレクトリの修正

RHBase ディレクトリ内のファイルの修正を行います.

①mclean，mk，minst の修正（B）

PCinRH と同様に修正を行います.

```
$cd /opt/RH/RHBase/src/comp
$cp mclean mclean.org
$cp mk mk.org
$cp minst minst.org
$gedit mclean mk minst
```

ここで，3 つのファイルとも，CollisionDetection，ObstacleAvoidance，ObstacleMonitor，RH2 の部分を削除して下さい．例として mclean を載せます.

「mclean」

```
---------------------------------------------------------------------
cd LocalizeCenter
make -f Makefile.LocalizeCenter clean
cd ..

cd Navigation
make -f Makefile.Navigation clean
cd ..

cd Odometry
make -f Makefile.Odometry clean
cd ..

cd PathFollower
make -f Makefile.PathFollower clean
cd ..

cd SwitchInput
make clean
cd ..
---------------------------------------------------------------------
```

3 つのファイルで該当部分を削除したら，修正は終了です.

②LocalizeCenter ディレクトリ内の修正

$cd /opt/RH/RHBase/src/comp/LocalizeCenter

Makefile.LocalizeCenter の修正を行います.

$cp Makefile.LocalizeCenter Makefile.LocalizeCenter.org

$gedit Makefile.LocalizeCenter

63 行目の下に追加（66 行目の上に追加）（D）

```
# add start
install:
        cp -p LocalizeCenterComp ../../../bin/comp
        cp -p LocalizeCenter.so ../../../bin/so
# add end
```

次に，LocalizeCenter.cpp の修正を行います．OdometryRTC からの入力データのみを受け取るように修正します.

$cp LocalizeCenter.cpp LocalizeCenter.cpp.org

$gedit LocalizeCenter.cpp

95 行目～103 行目の修正（B）

```
        m_odm.data.position.x = m_avPos.x = m_start_x;
        m_odm.data.position.y = m_avPos.y = m_start_y;
        m_odm.data.heading = m_avPos.theta = m_start_theta;

        //重みの設定
        m_odmWgt = 19.0;
        m_totalWgt = m_odmWgt;
```

132 行目～148 行目の修正（B）

```
//      if( ! ( start_x_prev == m_start_x && start_y_prev == m_start_y &&
start_theta_prev == m_start_theta ) )
//      {
                pos_init_cnt = 0;
                start_x_prev = m_start_x;
                start_y_prev = m_start_y;
                start_theta_prev = m_start_theta;

                /*
                m_odm.x = m_ceil.x = m_avPos.x = m_start_x;
```

```
                m_odm.y = m_ceil.y = m_avPos.y = m_start_y;
                m_odm.theta = m_ceil.theta = m_avPos.theta = m_start_theta;
                */
                m_odm.data.position.x = m_avPos.x = m_start_x;
                m_odm.data.position.y = m_avPos.y = m_start_y;
                m_odm.data.heading = m_avPos.theta = m_start_theta;

//          }
```

174 行目～240 行目の修正（OnExecute 内の修正）（B）

```
RTC::ReturnCode_t LocalizeCenter::onExecute(RTC::UniqueId ec_id)
{
        int odmFlag=0;

        //onActivate で位置を変えるときの処理．はじめに数回位置推定モジュールに
        //変更後の位置情報を流し込む.
        if(pos_init_cnt < 20){
                m_dp_out0Out.write();
                pos_init_cnt ++;
        }
        else{
                if(m_odmIn.isNew()){
                        while(!m_odmIn.isEmpty())//H.T  20100822  リングバッファの
最新の値を読む
                        {
                                m_odmIn.read();
                                odmFlag = 1;
                        }
                        m_avPos.x = m_odm.data.position.x;
                        m_avPos.y = m_odm.data.position.y;
                        m_avPos.theta = m_odm.data.heading;

                        if (m_avPos.theta > M_PI)                    //  180  度を越えて
いる場合、マイナスに変更する
                                m_avPos.theta = -M_PI * 2 + m_avPos.theta;
                        else if (m_avPos.theta < -M_PI)    //  -180  を下回る場合、プラ
```

スに変更する

```
                                        m_avPos.theta = M_PI * 2 + m_avPos.theta;

                                m_dp_out0.data.position.x = m_avPos.x;
                                m_dp_out0.data.position.y = m_avPos.y;
                                m_dp_out0.data.heading = m_avPos.theta;

                                m_dp_out0Out.write();
                                printf("%d                :                X=%f                Y=%f
Th=%f¥n",odmFlag,m_avPos.x,m_avPos.y,m_avPos.theta);
                        }
                }

   return RTC::RTC_OK;
}
```

③Odometry ディレクトリ内の修正

```
$cd /opt/RH/RHBase/src/comp/Odometry
```

Makefile.Odometry の修正を行います.

```
$cp Makefile.Odometry Makefile.Odometry.org
$gedit Makefile.Odometry
```

77 行目の上に追加（75 行目の下に追加）（D）

```
# add start
install:
        cp -p OdometryComp ../../../bin/comp
        cp -p Odometry.so ../../../bin/so
# add end
```

次に，Odometry.cpp の修正を行います.

```
$cp Odometry.cpp Odometry.cpp.org
$gedit Odometry.cpp
```

462 行目の「* 6.28/(6.28 - 0.2)」を削除（係数調整を削除）（D）

```
        dfai = Current.BodyOmega * timeStep;
```

④PathFollower ディレクトリ内の修正

$cd /opt/RH/RHBase/src/comp/PathFollower

PathFollower.cpp の修正を行います.

$cp PathFollower.cpp PathFollower.cpp.org

$gedit PathFollower.cpp

118 行目，120 行目の修正（*1.5 の削除）（B）

```
m_velocity.data.vx = target_v;
m_velocity.data.vy = 0;
m_velocity.data.va = target_w;
```

⑤SwitchInput ディレクトリ内の修正

$cd /opt/RH/RHBase/src/comp/SwitchInput

SwitchInputRTC.cpp の修正を行います. 自律移動のための目標速度の入力データのみを受け取るようにします.

$cp SwitchInputRTC.cpp SwitchInputRTC.cpp.org

$gedit SwitchInputRTC.cpp

65 行目～100 行目の修正（67 行目から 78 行目までの if 文の削除）（B）

```
RTC::ReturnCode_t SwitchInputRTC::onExecute(RTC::UniqueId ec_id)
{
if (m_inVelAutoIn.isNew())
    {
      m_inVelAutoIn.read();
      gettimeofday(&tv1,0);

      dsec = tv1.tv_sec;// - tv0.tv_sec;
      dusec = tv1.tv_usec;// - tv0.tv_usec;
      dusec = dsec*1000*1000+dusec;
      //printf("---------- %d¥n", dusec);

      if(dusec>500000){
        RTC_INFO(("AUTO vx:%f vy:%f TimeStamp: %ld[s] %ld[usec]",
m_inVelAuto.data.vx, m_inVelAuto.data.vy, tv1.tv_sec, tv1.tv_usec));
        std::cout << "AUTO vx: " << m_inVelAuto.data.vx << "vy: " <<
m_inVelAuto.data.vy << "AUTO va: " << m_inVelAuto.data.va << "TimeStamp: " <<
tv1.tv_sec << "[s] " << tv1.tv_usec << "[usec]" << std::endl;
        m_outVel.data.vx = m_inVelAuto.data.vx;
```

```
            m_outVel.data.vy = m_inVelAuto.data.vy;
            m_outVel.data.va = m_inVelAuto.data.va;
            m_outVelOut.write();
        }
    }
  usleep(1000);

  return RTC::RTC_OK;
}
```

⑤　～.conf の修正

～～.conf ファイルの修正を行います．修正を行うファイルは，LocalizeCenter.conf, Navigation.conf， NavigationParam.conf， Odometry.conf， OdometryParam.conf， PathFollower.conf, SwitchInput.conf です．LocalPosition.conf も修正を行いますが，地図作成に関連しているため 3.6 章で別途解説します．

```
$cd /opt/RH/RHBase/etc/conf
$gedit ./*.conf
```

まず，LocalizeCenter.conf，Navigation.conf，Odometry.conf，PathFollower.conf，SwitchInput.conf において，

```
logger.file_name:/opt/rh/RS003/log/**.log
```

を，

```
logger.file_name:/opt/RH/RHBase/log/**.log
```

とします．（B）

また，Navigation.conf，Odometry.conf，PathFollower.conf，SwitchInput.conf において，

```
exec_cxt.periodic.rate: 50.0
```

を，

```
exec_cxt.periodic.rate: 1000.0
```

とします．（B）

また，OdometryParam.conf において，最後に

```
## ThreeWheeledRobot [Simulation] ##
conf.ThreeWheeledRobot.leftWheelID: 1
conf.ThreeWheeledRobot.rightWheelID: 0
conf.ThreeWheeledRobot.radiusOfLeftWheel: 0.018
conf.ThreeWheeledRobot.radiusOfRightWheel: 0.018
conf.ThreeWheeledRobot.lengthOfAxle: 0.078
```

conf.ThreeWheeledRobot.radiusOfBodyArea: 0.15
を付け加えて下さい．（C）
　これは，3 輪移動ロボットのシミュレーションモデルの値です．
　また，NavigationParam.conf において，最後に
## ThreeWheeledRobot ##
conf. ThreeWheeledRobot.judge_radius: 0.15
conf. ThreeWheeledRobot.max_vel: 0.4
を付け加えて下さい．（C）

（3）RemotePC ディレクトリの修正
　RemotePC ディレクトリ内のファイルの修正を行います．
①mclean，mk，minst の修正（B）
　PCinRH，RHBase と同様に修正を行います．
$cd /opt/RH/RemotePC/src/comp
$cp mclean mclean.org
$cp mk mk.org
$cp minst minst.org
$gedit mclean mk minst
　ここで，3 つのファイルとも，DispPosition，PositionInput，PathPlanning 以外の部分を削除して下さい．例として mclean を載せます．
「mclean」
---
cd ./DispPosition
make -f Makefile.DispPosition clean
cd ..

cd ./PositionInput
make -f Makefile.PositionInput clean
cd ..

cd ./PathPlanning
make -f Makefile.PathPlanning clean
cd ..
---

②DispPosition ディレクトリ内の修正

DispPosition.h と DispPosition.cpp の修正を行いますが，これは地図作成に関係しています．よって，3.6.4 章で別途解説を行います．

③PathPlanning ディレクトリ内の修正

planner.cpp の修正を行いますが，これは地図作成に関係しています．よって，3.6.4 章で別途解説を行います．

④　～.conf の修正

～～.conf ファイルの修正を行います．修正を行うファイルは，DispPosition.conf，PathPlanning.conf，PathPlanningParam.conf，PositionInput.conf です．Position.conf も修正を行いますが，地図作成に関連しているため 3.6.4 章で別途解説します．

$cd /opt/RH/RemotePC/etc/conf

$gedit DispPosition.conf PathPlanning.conf PathPlanningParam.conf PositionInput.conf

まず，DispPosition.conf，PathPlanning.conf，PositionInput.conf において，

logger.file_name:/opt/rh/RS003/log/**.log

を，

logger.file_name:/opt/RH/RemotePC/log/**.log

とします．（B）

また，PathPlanningParam.conf において，最後に

## ThreeWheeledRobot ##

conf.ThreeWheeledRobot.map_file: /opt/RH/RemotePC/bin/comp/route_map

を追加します．（C）

### ３．６　経路・経由点が記してある地図の作成

#### ３．６．１　実環境に即した地図の作成（A）

実環境に即した地図画像の作成を行います．

現実に即したシミュレーション環境を作成するために，まず，ロボットを走行させるフィールドや障害物などのサイズを全てメジャーなどで計測して下さい．この時，ロボットの走行経路や経由点，初期位置をおおよそ把握することを推奨します．

次に，例（図 3.6.1）のように実環境を簡略化した地図画像を作成して下さい．地図の色の組み合わせは，白黒以外でも構いません．この際，必要ならば壁も障害物の一種と考え，地図に書き入れて下さい．この地図画像は「アプリケーション＞オフィス＞OpenOffice.org Presentation」にて，長方形や円を組み合わせて，最後にグループ化を行うことで作成することができます．よって，まず，「アプリケーション＞オフィス」にて，「OpenOffice.org Presentation」があることを確認して下さい．ない場合は，OpenOffice

のインストールを行って下さい.

OpenOffice.org Presentation での保存は，グループ化した画像を右クリックし「変換＞ビットマップに変換」を選択，その後もう一度右クリックし，「図として保存」をクリックするという方法となっております.

この画像は適度な大きさ（目安としては縦横 700 ピクセル以内で，障害物とそうでない場所の区別がはっきりできる程度）で作成し，保存時に「JPEG - Joint Photographic ExpertsGroup」を選択し/opt/RH/RemotePC/bin/comp/に，また，保存時に「BMP - Windows Bitmap」を選択し/opt/RH/RemotePC/src/tool/MapManager/src/に保存して下さい．それぞれ任意の名称で構いません.

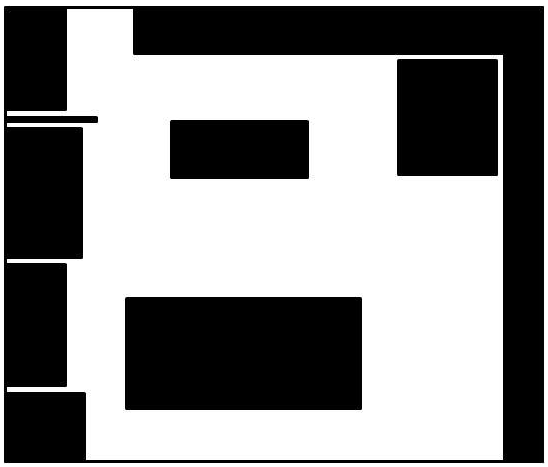

図 3.6.1　地図画像例

### ３．６．２　MapEditor.java の修正

「MapEditor.java」というファイルの修正を行います．このファイルの修正を行うことで，本学習環境にて使用する，ロボットが移動する際の経路と経由点が記されている地図を作成するためのツール「MapManager」を使用できるようになります.

まず，端末を起動（「アプリケーション＞アクセサリ＞端末」）し，以下に移動します.

$cd /opt/RH/RemotePC/src/tool/MapManager/src

このとき，キーボード上の「tab」キーを活用することで時間の節約が行える場合があります．例えば，

$cd /opt/RH/Re

まで入力しているときにキーボードの「tab」キーを押下すると，

$cd /opt/RH/RemotePC/

まで出現します．このように，同じディレクトリ内で類似した名称のファイルやディレクトリが無い場合，「tab」キーによって自動的に名称の続きを出現させることができます．今後様々なファイルの修正を行う際，「tab」キーを活用することで時間の短縮を行えるようになる場合があるため，積極的に活用して下さい.

それでは修正作業に戻り，MapEditor.java の修正を行います.

$gedit MapEditor.java

・501 行目を編集（C）

500 行目 ：　　try {

501 行目 ：　　readImage = ImageIO.read(new File("509map.bmp"));

502 行目 ：　　imgWidth = readImage.getWidth();

にて，“509map.bmp”を自分の保存したファイル名にします.

・515 行目を編集（C）

514 行目 ：　　scale = 1.0;

515 行目 ：　　mPerPix = 0.0132; //1pixel あたり 1.32cm

516 行目 ：　　setOrigin(-271 , -230);

にて，mPerPix = 0.0132;　を，自分の bmp の地図の 1 ピクセル当りの長さ[m]に変換します．これは，「実際にメジャーなどで計測した長さ」÷「bmp 画像を開いた際にウインドウ左下に出るピクセル数」を計算することで求めることができます. 自分の bmp の地図の縦・横どちらで計算しても構いません.

・516 行目を編集（C）

515 行目 ：　　mPerPix = 0.0132; //1pixel あたり 1.32cm

516 行目 ：　　setOrigin(-271 , -230);

517 行目 ：　　setBackground(Color.gray);

にて，setOrigin(-271,-230);の数字を，自分の bmp の地図において設定したい原点の座標に置き換えます．bmp 画像の画面右下隅が原点(0,0)で，下方向が Y+，右方向が X+，座標の単位はピクセル数です．ピクセル数であるため，数値は整数で設定して下さい．数字の意味は図 3.6.2 に示した通りとなっております.

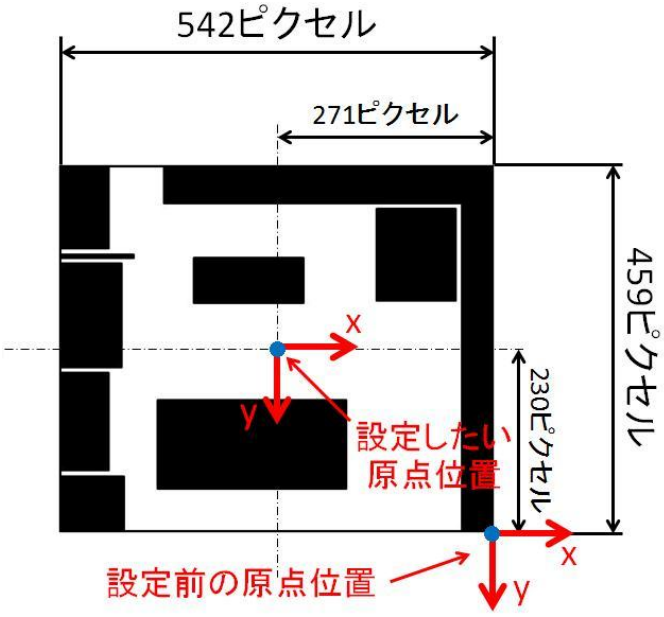


図 3.6.2　MapEditor.java の修正のための参考図

次にコンパイルをします．現状では，このディレクトリで作業を行うため，すべての java ファイルをコンパイルします．

$javac Links.java MapEditor.java MapManager.java Nodes.java

これで，MapManager を実行するための準備は完了です．

### ３．６．３　MapManager の実行

3.6.2 章のディレクトリに移動し，MapManager を実行します．

$cd /opt/RH/RemotePC/src/tool/MapManager/src

$java MapManager

そして，

/opt/RH/RemotePC/src/tool/MapManager/src/MapEditorManual.pptx

と，

来訪者受付システム Ver1.0 でダウンロードできる「オープンソース移動知能モジュール群 経路計画・軌道追従モジュール機能仕様書」の 3.5 章の解説の通りに，Node と Link を設定します．Node が経由点や出発地点，目標地点となり，Link が経路となります．Node の設定において，nodeID が MapManager 側で設定する Node の ID で，Node を作成した順に 0 から ID が割り振られます．また，X，Y，THETA が図 3.6.3 に示す座標系における座標と角度となっております．THETA が 0[rad]の場合ロボットは X 軸正方向向き（右向き）が前方となり，THETA が 1.57[rad]の場合ロボットは Y 軸正方向向き（上向き）が前方となります．

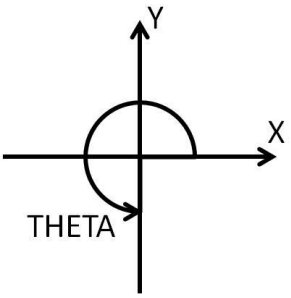


図 3.6.3　MapManager の座標系

そして，「FILE＞SAVE」で route.dat などのファイル名で dat ファイルを任意の位置に保存します．これは，再度この経路地図を編集するときに必要なファイルです．「FILE＞OPEN」よりこの dat ファイルを開くことで，再編集が可能となります．

また，「FILE＞Write RouteMap」で route_map というファイル名で，/opt/RH/RemotePC/bin/comp/に保存します．

視覚的にも情報を残すために，この経路のスクリーンショットを撮ります．キーボード上の「Alt を押しながら Print Screen を押下」（アクティブなウインドウのみのスクリーンショット）や「Print Screen を押下」（デスクトップ全体のスクリーンショット）を行ってスクリーンショットを撮り，route_map.png というファイル名で /opt/RH/RemotePC/bin/comp/に保存して下さい．route_map.png の例を図 3.6.4 に示します．

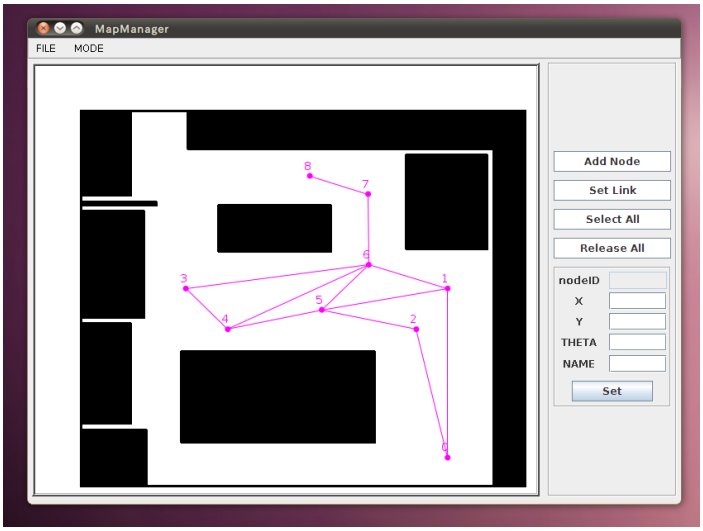


図 3.6.4　route_map.png の例

これで経路地図作成は終了です．MapManager を終了して下さい．

### ３．６．４　各ファイルの修正

（１）DispPosition.cpp

DispPosition ディレクトリに移動し，DispPosition.cpp の編集を行います．

$cd /opt/RH/RemotePC/src/comp/DispPosition

$gedit DispPosition.cpp

・75 行目を編集（C）

mPerPix を，3.6.2 章にて MapEditor.java で修正したように自分のファイルの 1 ピクセル当りの長さ[m]に修正します．

・108 行目，109 行目を編集（C）

106 行目：　　rth = M_PI * 2 + rth;

107 行目：

108 行目；　　rx = round( m_pos.data.position.x / mPerPix ) + 271;

109 行目：　　ry = - round( m_pos.data.position.y / mPerPix ) + (459 - 230);

110 行目：

111 行目：　　//ロボット位置の描画

にて，「…+ 271;」と「…+ (459 - 230);」を修正します．この数字部分は，地図画像における原点位置の指定であり，画像に合わせて原点の位置や座標軸の方向に注意して修正する必要があります．数字の意味は図 3.6.5 に示した通りとなっております．図に示す通りこの数値はピクセル数であるため，整数で設定して下さい．

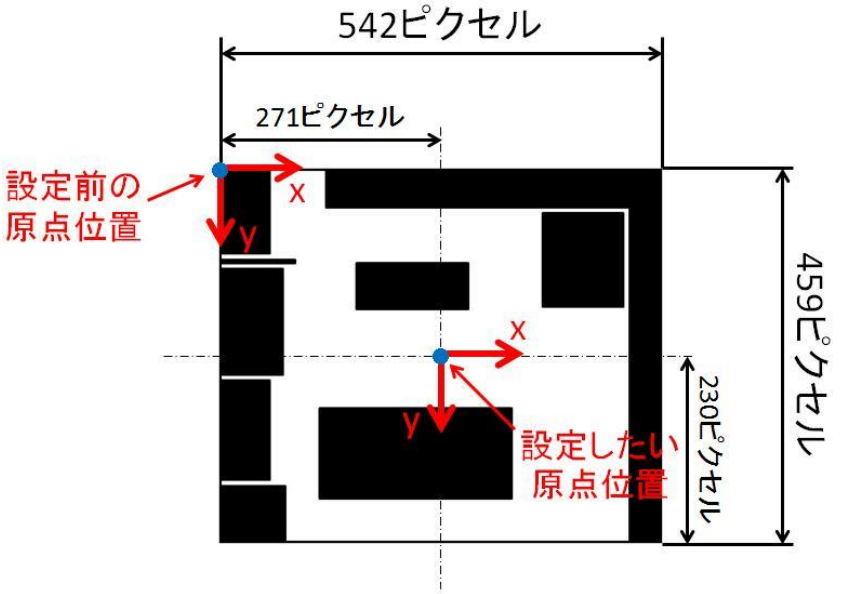


図 3.6.5　DispPosition.cpp の修正のための参考図

（２）DispPosition.h

DispPosition ディレクトリに移動し，DispPosition.h の編集を行います.

$cd /opt/RH/RemotePC/src/comp/DispPosition

$gedit DispPosition.h

・25 行目を編集（C）

23 行目：　　#include "highgui.h"

24 行目：

25 行目：#define ORG_MAP "509map.jpg"

26 行目：

27 行目：using namespace RTC;

にて，jpg ファイルの名前を自分で保存したファイル名に変更して下さい.

（３）PathPlanning 内の planner.cpp

PathPlanning ディレクトリに移動し，planner.cpp の編集を行います.

$cd /opt/RH/RemotePC/src/comp/PathPlanning

$gedit planner.cpp

532 行目，533 行目を編集（C）

…[-16:16]　→　[-8:8]に変更

…[-12:12]　→　[-6:6]に変更

これは実際に RT ミドルウェア学習環境を実行した際，図 3.6.6 のような画像の x の範囲と y の範囲を決定するパラメータです．よって，実際に RT ミドルウェア学習環境を実行した後，経路が小さく表示された場合や大きくて画面内に表示しきれない場合などに適宜調整して下さい.

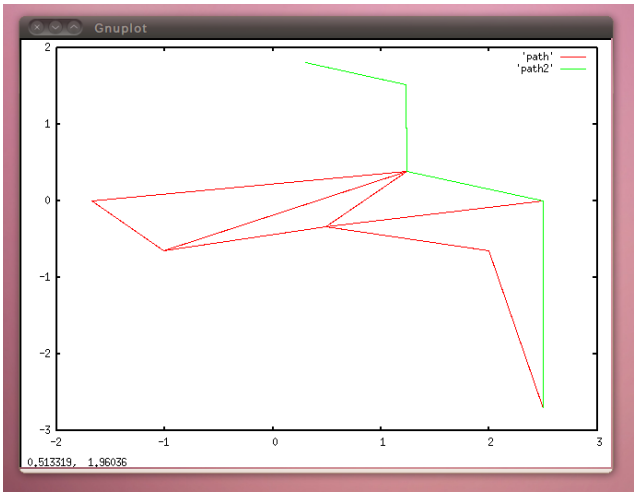


図 3.6.6　経路画像の例

（4）RHBase 内の LocalPosition.conf

RTC「LocalizeCenter」で使用する LocalPosition.conf の修正を行います.

まず，MapManager 実行時に作成した route_map.png を開きます.

$cd /opt/RH/RemotePC/bin/comp/

$eog route_map.png

また，MapManager 実行時に作成した route_map も別の端末にて開きます.

$cd /opt/RH/RemotePC/bin/comp/

$gedit route_map

次に，編集するファイルである LocalPosition.conf を開きます．RemotePC 内にも同名のファイルが存在しますので間違えないよう注意して下さい.

$cd /opt/RH/RHBase/etc/conf

$gedit LocalPosition.conf

これで，LocalPosition.conf のフォーマットを崩さないようにしながら，route_map と route_map.png を参考に自分の設定した経路地図座標を入力して下さい.

以下に，編集方法を示します．数値はあくまで例ですので，各地図に合わせて変更して下さい.

まず，図 3.6.7 の route_map.png の場合，route_map は，以下の内容となっています.

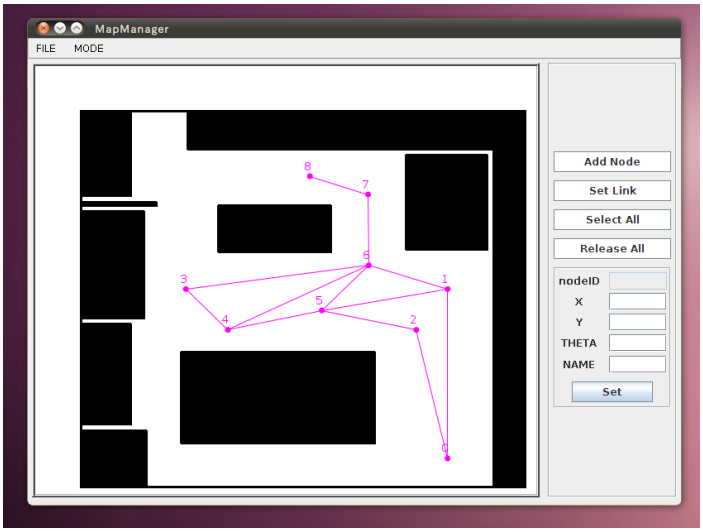


図 3.6.7　route_map.png の例

・route_map

---

n 2.5 -2.7 3.14

n 2.5 0.0 3.14

n 2.0 -0.65 1.57

n -1.68 0.0 0.0

n -1.01 -0.65 1.57
n 0.49 -0.34 3.14
n 1.24 0.38 1.57
n 1.23 1.51 2.62
n 0.3 1.8 3.14
l 0 1 0
l 0 2 0
l 2 5 0
l 1 6 0
l 6 7 0
l 7 8 0
l 3 4 0
l 4 5 0
l 3 6 0
l 4 6 0
l 1 5 0
l 5 6 0
---

n から始まる行は Node を表しており，l（える）から始まる行は Link を表しています．

n 2.5 -2.7 3.14

ならば，図 3.6.8 の座標系（図 3.6.3 と同図）の通り，X 座標が 2.5[m]，Y 座標が-2.7[m]，THETA が 3.14[rad]であることを表しています．また，n で始まる行の最初が NodeID0 であり，順に 1，2，3，…と，この例では計 9 個の Node があることがわかります．

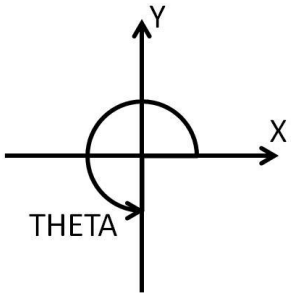


図 3.6.8　MapManager の座標系

l 0 1 0

ならば，NodeID0 と NodeID1 の間に Link（経路）が設定されていることを表しています．なお，これは「える・ぜろ・いち・ぜろ」であり，l（える）と 1（いち）の見間違いに注意して下さい．

Node は図 3.6.9 のような関係となっております．図 3.6.9 の Node ごとの表記は，NodeID(X,Y,THETA)という形式で表されています．

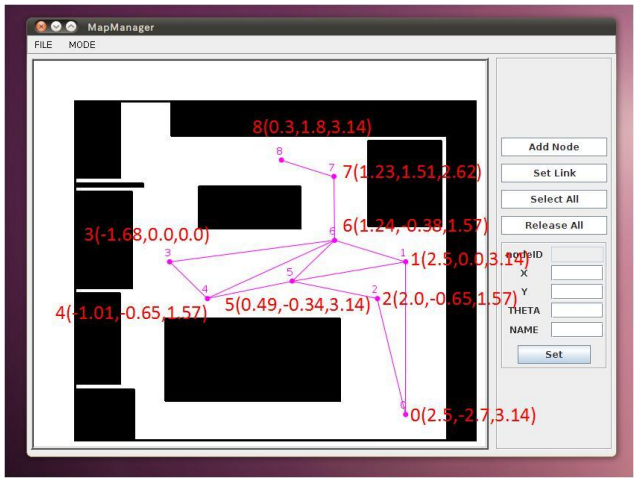


図 3.6.9　route_map と route_map.png の関係

これに対応させるため，LocalPosition.conf は以下の内容にします．

・LocalPosition.conf

---

```
port.inport.dp_ceilIn.buffer.length: 1
port.inport.dp_ceilIn.buffer.read.empty_policy: block

conf._widget_.startX: text
conf._widget_.startY: text
conf._widget_.startTheta: text

## default ##
conf.default.startX: 2.50
conf.default.startY: -2.70
conf.default.startTheta: 3.14

## 0 ##
conf.0.startX: 2.50
conf.0.startY: -2.70
conf.0.startTheta: 3.14

## 1 ##
conf.1.startX: 2.50
```

conf.1.startY: 0.00
conf.1.startTheta: 3.14

## 2 ##
conf.2.startX: 2.0
conf.2.startY: -0.65
conf.2.startTheta: 1.57

## 3 ##
conf.3.startX: -1.68
conf.3.startY: 0.00
conf.3.startTheta: 0.00

## 4 ##
conf.4.startX: -1.01
conf.4.startY: -0.65
conf.4.startTheta: 1.57

## 5 ##
conf.5.startX: 0.49
conf.5.startY: -0.34
conf.5.startTheta: 3.14

## 6 ##
conf.6.startX: 1.24
conf.6.startY: 0.38
conf.6.startTheta: 1.57

## 7 ##
conf.7.startX: 1.23
conf.7.startY: 1.51
conf.7.startTheta: 2.62

## 8 ##
conf.8.startX: 0.3
conf.8.startY: 1.8

conf.8.startTheta: 3.14

---

まず,

---

port.inport.dp_ceilIn.buffer.length: 1

port.inport.dp_ceilIn.buffer.read.empty_policy: block

conf._widget_.startX: text

conf._widget_.startY: text

conf._widget_.startTheta: text

---

は内容を変更しないで下さい．次に,

---

## default ##

conf.default.startX: 2.50

conf.default.startY: -2.70

conf.default.startTheta: 3.14

---

は，NodeID0 の X 座標，Y 座標，THETA を入れて下さい．また,

---

## 0 ##

conf.0.startX: 2.50

conf.0.startY: -2.70

conf.0.startTheta: 3.14

---

以降は,

---

## NodeID ##

conf.NodeID.startX: その NodeID における X 座標

conf.NodeID.startY: その NodeID における Y 座標

conf.NodeID.startTheta: その NodeID における THETA

---

という書式で，自分が作成した地図の Node 数だけ情報を追加して下さい．これで,

LocalPosition.conf の編集は終了です.

（５）RemotePC 内の Position.conf

（４）と同じ手順で RTC「PositionInput」で使用する Position.conf の修正を行います.

まず，MapManager 実行時に作成した route_map.png を開きます.

```
$cd /opt/RH/RemotePC/bin/comp/
$eog route_map.png
```

また，MapManager 実行時に作成した route_map も別の端末にて開きます.

```
$gedit route_map
```

次に，編集するファイルである Position.conf を開きます.

```
$cd /opt/RH/RemotePC/etc/conf
$gedit Position.conf
```

これで，Position.conf のフォーマットを崩さないようにしながら，route_map と route_map.png を参考に，自分の設定した経路地図座標を入力して下さい.

以下に編集方法を示します. route_map と route_map.png については LocalPosition.conf と同様です．それらに対応させるため，Position.conf の内容は以下のようにします.

・Position.conf

```
-------------------------------------------------
conf._widget_.01_x: text
conf._widget_.02_y: text
conf._widget_.03_th: text

## default ##
conf.default.01_x: 2.50
conf.default.02_y: -2.70
conf.default.03_th: 3.14

## 0 ##
conf.0.01_x: 2.50
conf.0.02_y: -2.70
conf.0.03_th: 3.14

## 1 ##
conf.1.01_x: 2.50
conf.1.02_y: 0.00
conf.1.03_th: 3.14

## 2 ##
```

conf.2.01_x: 2.0
conf.2.02_y: -0.65
conf.2.03_th: 1.57

## 3 ##
conf.3.01_x: -1.68
conf.3.02_y: 0.00
conf.3.03_th: 0.00

## 4 ##
conf.4.01_x: -1.01
conf.4.02_y: -0.65
conf.4.03_th: 1.57

## 5 ##
conf.5.01_x: 0.49
conf.5.02_y: -0.34
conf.5.03_th: 3.14

## 6 ##
conf.6.01_x: 1.24
conf.6.02_y: 0.38
conf.6.03_th: 1.57

## 7 ##
conf.7.01_x: 1.23
conf.7.02_y: 1.51
conf.7.03_th: 2.62

## 8 ##
conf.8.01_x: 0.3
conf.8.02_y: 1.8
conf.8.03_th: 3.14
---
まず,
---

conf._widget_.01_x: text
conf._widget_.02_y: text
conf._widget_.03_th: text
---
は内容を変更しないで下さい．次に，
---
## default ##
conf.default.01_x: 2.50
conf.default.02_y: -2.70
conf.default.03_th: 3.14
---
は，NodeID0 の X 座標，Y 座標，THETA を入れて下さい．また，
---
## 0 ##
conf.0.01_x: 2.50
conf.0.02_y: -2.70
conf.0.03_th: 3.14
---
以降は，
---
## NodeID ##
conf.NodeID.01_x: その NodeID における X 座標
conf.NodeID.02_y: その NodeID における Y 座標
conf.NodeID.03_th: その NodeID における THETA
---
という書式で，自分が作成した地図の Node 数だけ情報を追加して下さい．これで，Position.conf の編集は終了です．

以上で，経路データの修正は完了です．

なお，3.6 章で作成したファイルの例として，著者の作成したファイル群をダウンロードすることができます．ホームページにてダウンロードできる「ユーザマニュアル関連情報.zip」内の「3.6 章」というディレクトリをご覧下さい．

## ３．７　シミュレーション環境の構築

### ３．７．１　シミュレーションモデルの作成（A）

3.6.1 章で作成した地図画像に基づき，OpenHRP3 で使用するシミュレーションモデル（VRML ファイル，拡張子は wrl）を作成して下さい．

モデルの作成方法に関しては，OpenHRP3 のユーザーズマニュアル（Ver.3.1.1）にあるモデル作成ガイド内の，

VRML モデルの概要：http://www.openrtp.jp/openhrp3/jp/create_model.html

GrxUI を用いたモデル作成：http://www.openrtp.jp/openhrp3/jp/howtoeditmodel.html

にて解説されております．

また，3 輪移動ロボットのシミュレーションモデルと，障害物のシミュレーションモデル作成のための雛形となるモデル（VRML ファイル）として，床モデル，直方体モデル，円柱モデルは著者によって用意されております．それらはホームページにてダウンロードできる「ユーザマニュアル関連情報.zip」内にあります．これを使用する場合，全てのモデルについて，モデルの位置とサイズを変更するのみで障害物モデルを作成することができるため，以下の手順でシミュレーション環境を容易に構築することができます．なお，長さの単位は全て[m]です．

まず，ダウンロードした SimulationModel の中に，RT ミドルウェア学習環境にて用いるシミュレーションモデルがあります．端末を開き，シミュレーションモデルを以下のコマンドで指定する場所へコピーします．半角スペースに注意して入力して下さい．

```
$sudo cp -p -r (解凍した場所)/SimulationModel/ThreeWheeledRobot
/usr/share/OpenHRP-3.1/sample/model/
```

それではまず，床モデルを修正する方法です．床モデルはサイズのみの修正です．端末を開き，VRML ファイルを以下のように修正します．

```
$cd /usr/share/OpenHRP-3.1/sample/model/ThreeWheeledRobot/EnvironmentModel
$gedit floor01.wrl
```

・5 行目を編集

4 行目：　　geometry Box {

5 行目：　　size 6.04 7.14 0.020

6 行目：　　}

にて，6.04 が x，7.14 が y，0.020 が z の値です．この内，x と y の値を自分のフィールドに合わせて修正して下さい．

なお，床モデルの x 軸，y 軸は図 3.7.1 のように定義されております．直方体モデル，円柱モデルの作成時にも参考にして下さい．

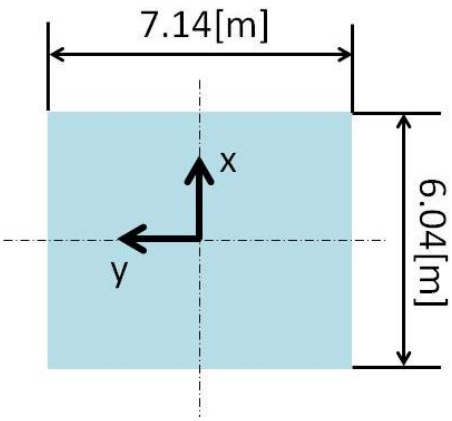


図 3.7.1　床モデルの xy 軸

次に，直方体モデルを修正する方法です．まず直方体モデルの場所まで移動し，直方体モデルをコピーして OBSTACLE_BOX01.wrl という VRML ファイルを作成します．そして，それを修正します．障害物のモデルを増やす場合は，以下の手順において「OBSTACLE_BOX01」を，「OBSTACLE_BOX02」，「OBSTACLE_BOX03」，……，などというように，モデルごとに番号を変えて作成して下さい．

では，端末を開き，以下を入力します．

$cd /usr/share/OpenHRP-3.1/sample/model/ThreeWheeledRobot/EnvironmentModel

$cp -p OBSTACLE_BOX.wrl OBSTACLE_BOX01.wrl

$gedit OBSTACLE_BOX01.wrl

・203 行目を編集

202 行目：　　Transform {

203 行目：　　translation 0.000 0.000 0.000

204 行目：　　rotation 0 1 0 0

にて，床モデルにおける図 3.7.1 の軸に従い，3.6.1 章で計測した実環境と同様の配置となるよう x，y の値を変更して下さい．この座標はシミュレーションモデルの中心の座標を表しています．

・208 行目を編集

207 行目：　　geometry Box{

208 行目：　　size 1.000 1.000 0.500

209 行目：　　}

にて，これも床モデルにおける図 3.7.1 の軸に従い，3.6.1 章で計測した実環境と同様のサイズとなるよう x，y の値を変更して下さい．なお，高さは 0.500[m]のまま変更しないで下さい．

・190，192，197，224，227 行目を編集
OBSTACLE_BOX の後に，OBSTACLE_BOX01 や OBSTACLE_BOX02 などというように番号を追加して下さい．

次に，円柱モデルを修正する方法です．大まかな流れは直方体モデルと同様です．
$cd /usr/share/OpenHRP-3.1/sample/model/ThreeWheeledRobot/EnvironmentModel
$cp -p OBSTACLE_CYLINDER.wrl OBSTACLE_ CYLINDER01.wrl
$gedit OBSTACLE_ CYLINDER01.wrl
・203 行目を編集
202 行目：　　Transform {
203 行目：　　tranlation 0.000 0.000 0.000
204 行目：　　rotation 1 0 0 1.5708
にて，床モデルにおける図 3.7.2（図 3.7.1 と同図）の軸に従い，3.6.1 章で計測した実環境と同様の配置となるよう x，y の値を変更して下さい．

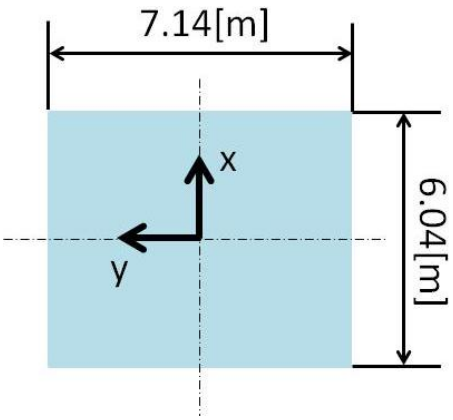


図 3.7.2　床モデルの xy 軸

・208 行目を編集
207 行目：　　geometry Cylinder{
208 行目：　　radius 1.000
209 行目：　　height 0.500
にて，「radius 1.000」は円柱モデルの底面の半径の長さなので，3.6.1 章で計測した実環境と同様のサイズとなるよう値を変更して下さい．

・190，192，197，225，228 行目を編集
OBSTACLE_CYLINDER の後に，OBSTACLE_CYLINDER01 や
OBSTACLE_ CYLINDER02 などというように番号を追加して下さい．
以上で修正は終了です．3.6.1 章で計測した実環境と同数，同サイズの障害物を作成した

ことを確認して下さい．作成したモデルは，
/usr/share/OpenHRP-3.1/sample/model/ThreeWheeledRobot/EnvironmentModel 内に保存して下さい．

また，3 輪移動ロボットについてはシミュレーションモデルが既に作成されております．シミュレーションにおけるロボットモデルの初期位置（出発地点）と初期姿勢のみ変更する必要があるため，まず，3.6.3 章で設定した Node の中から出発地点を決定して下さい．

その後，端末を開き，ロボットモデルの VRML ファイルを以下のように変更します．

```
$cd /usr/share/OpenHRP-3.1/sample/model/ThreeWheeledRobot/RobotModel
$gedit MODEL_CAR.wrl
```

・212，213 行目を編集

211 行目：　　jointType "free"
212 行目：　　translation -2.7 -2.0 0.028
213 行目：　　rotation 0 0 1 3.14
214 行目：　　gearRatio 58.2

にて，「translation 0.000 0.000 0.028」は初期位置の指定，「rotation 0 0 1 0.00」は初期姿勢の指定をしています．図 3.7.3 の軸（図 3.7.1，図 3.7.2 と同図）に従って，3.6.3 章で設定した Node の値を変換し translation の x，y と rotation の最後の値に入力して下さい．
例えば Node の値が 1.0 -2.0 3.14 だった場合，
translation -2.000 -1.000 0.028
rotation 0 0 1 3.14
となります．

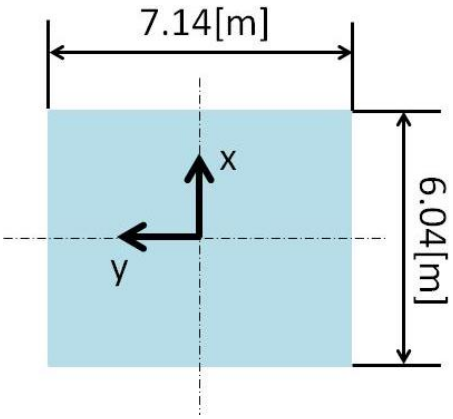


図 3.7.3　床モデルの xy 軸

また，用意したモデルを使用せず，1 から新しくシミュレーションモデルを作成した場合は，/usr/share/OpenHRP-3.1/sample/model/の中に作成する ThreeWheeledRobot などのディレクトリに VRML ファイルを保存すると実行時に手間が省けます．

これで，シミュレーションモデルの作成は終了です．

### ３．７．２　OpenHRP3 での設定（A）

Eclipse を起動します.

まず,「アプリケーション＞アクセサリ＞端末」で，端末を起動します．端末が起動したら，3.3 章の「Ubuntu9.10 以降の環境での Eclipse の起動」で作成したシェルスクリプト（script.sh）がある場所まで移動し，実行します.

$ cd /home/(アカウント名)/eclipse

$sudo sh script.sh

これで，Eclipse が起動します.

次に,「ウインドウ＞パースペクティブを開く＞その他」（図 3.7.4）で，GrxUI を選択し（図 3.7.5）実行します.

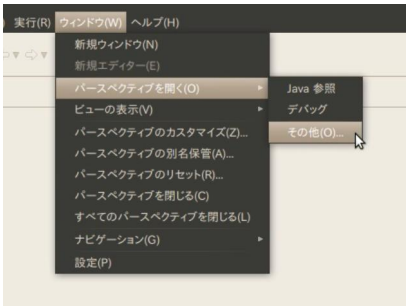


図 3.7.4　パースペクティブを開く

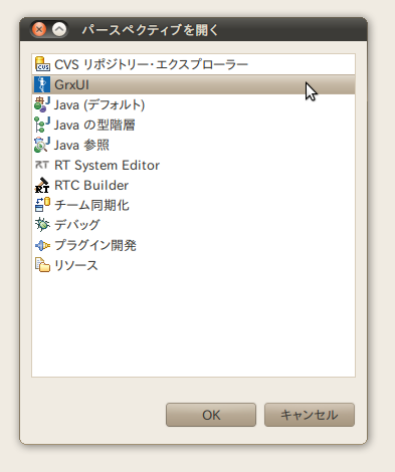


図 3.7.5　GrxUI の選択

これで GrxUI が開くはずですが，アイテムや 3D ビューなどが表示されないことがあります（図 3.7.6）．そこで，図 3.7.7 のように「ウインドウ＞新規ウインドウ」で新規ウインドウを開きます．新規ウインドウが開かれたら前のウインドウを終了することで，GrxUI が正常に実行可能となります．（図 3.7.8）

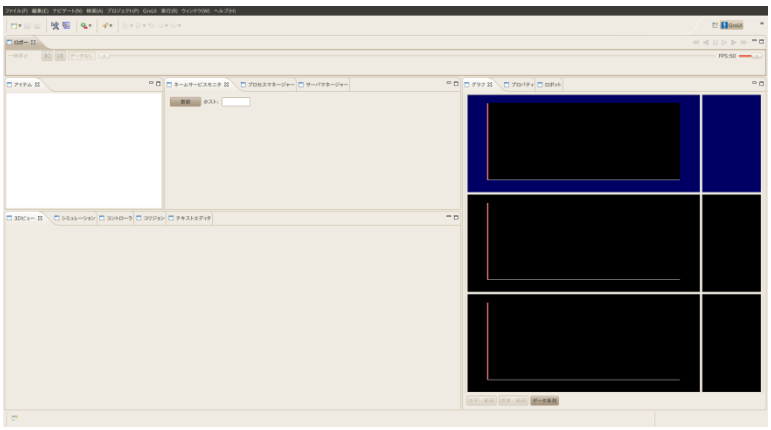

図 3.7.6　GrxUI でアイテムなどが表示されない場合


図 3.7.7　新規ウインドウを開く

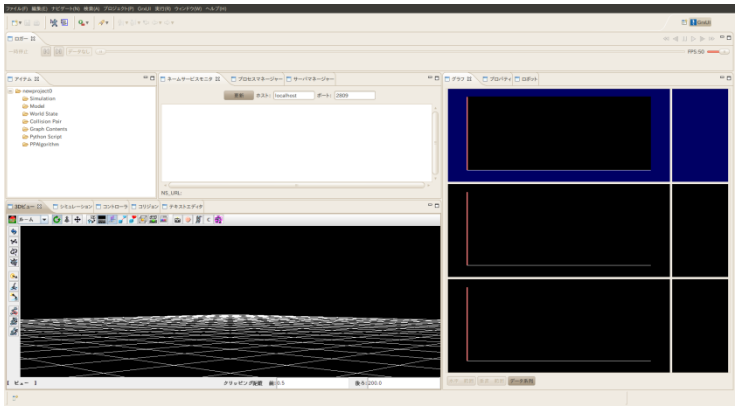

図 3.7.8　GrxUI でアイテムなどが表示されている場合

次に，プロジェクトの作成・保存を行います.

まず，図 3.7.9 のようにアイテムの newproject0 の上で右クリックし，「プロジェクトの作成」を選択して下さい. すると, 図 3.7.10 のようなウインドウが表示されますので,「OK」をクリックします.


図 3.7.9　プロジェクトの作成

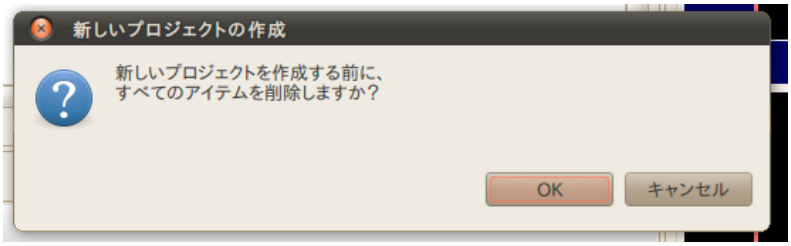


図 3.7.10　プロジェクトの新規作成に関する確認

次に, 図 3.7.11 のように「Simulation の上で右クリック＞作成」をし, 同様に「World State の上で右クリック＞作成」をして下さい．これで，newsimulation0 と newworldstate0 が作成されます.


図 3.7.11　Simulation の上で作成の選択

次に，「Model の上で右クリック＞読み込み」を選択し，3.7.1 章で作成したモデルを全て読み込んで下さい. 読み込むモデルは, 3 輪移動ロボットのモデルである MODEL_CAR, 床モデルである MODEL_FLOOR，また，作成した障害物モデル（OBSTACLE_BOX01 や OBSTACLE_CYLINDER01 など）です．この際，MODEL_CAR 以外は，図 3.7.12 に示す通り「環境モデルへ変更」を行い，図 3.7.12 の MODEL_FLOOR の左のアイコンに変更させて下さい.


図 3.7.12　環境モデルへ変更

すると，色や障害物の数，フィールドの広さなど違いはありますが，図 3.7.13 のようなシミュレーション環境が構築されます．図 3.7.13 は上から見た図で，かつ，3D ビューを最大化表示している表示方法です．

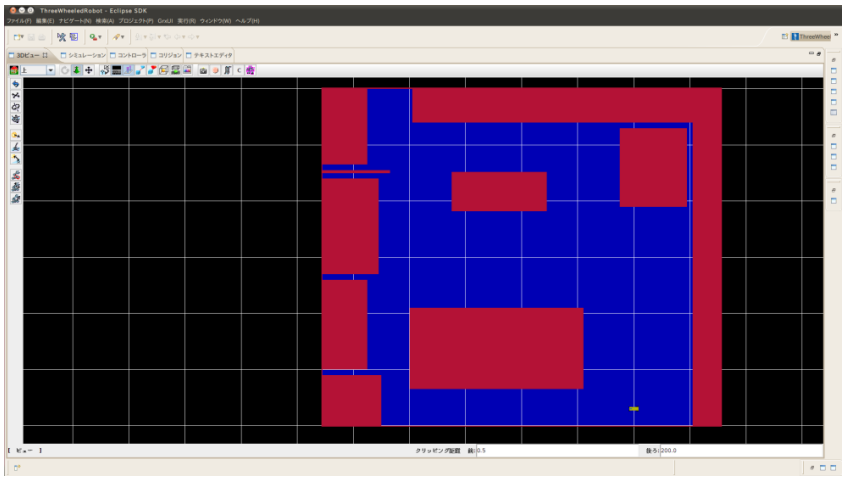

図 3.7.13　シミュレーション環境構築例

次に，左下のウインドウにある「シミュレーション」,「コリジョン」の設定を行います．

「シミュレーション」タブに移動して下さい．そこで，総時間は十分長く設定し（例えば 600），積分時間は 0.003，ログ時間[秒]は 0.1，積分方法は RUNGE_KUTTA，重力[m/s^2]は 9.8 とし，順動力学と実時間にチェックを入れて下さい．

「コリジョン」タブに移動して下さい．「コリジョン」は，「追加」をクリックすることで，オブジェクト同士の摩擦などの設定を行えるようになります．特に変更したい箇所がない場合，表 3.7.1 のように設定して下さい．なお，RT ミドルウェア学習環境では障害物回避を行うことができないため，3 輪移動ロボットのシミュレーションモデルと障害物のシミュレーションモデルとの間では衝突判定を行わないようにします．つまり，3.7.1 章で障害物として作成した VRML ファイルのコリジョンの設定は行わず，3 輪移動ロボットと障害物が衝突することはありません．

表 3.7.1　コリジョンの設定

| オブジェクト 1 | リンク 1 | オブジェクト 2 | リンク 2 | 静摩擦係数 | すべり摩擦係数 | 接触点選択幅 |
|---|---|---|---|---|---|---|
| MODEL_CAR | MAIN_BODY | MODEL_FLOOR | floor | 0.5 | 0.5 | 0.01 |
| MODEL_CAR | RIGHT_WHEEL | MODEL_FLOOR | floor | 0.5 | 0.5 | 0.01 |
| MODEL_CAR | LEFT_WHEEL | MODEL_FLOOR | floor | 0.5 | 0.5 | 0.01 |
| MODEL_CAR | CASTER_WHEEL | MODEL_FLOOR | floor | 0 | 0 | 0.01 |

これでプロジェクトの作成は終了しました．

次に，作成したプロジェクトを保存します．図 3.7.14 のように「GrxUI＞名前を付けてプロジェクトの保存」を選択し，「/usr/share/OpenHRP-3.1/sample/project/」内に「ThreeWheeledRobot.xml」など任意の名称で保存して下さい．

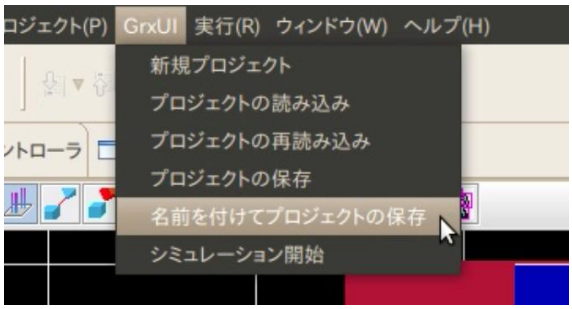


図 3.7.14　名前を付けてプロジェクトの保存

ここで，図 3.7.15 のように「ウインドウ設定を保存しますか？」と尋ねられます．現在 GrxUI にて表示しているウインドウの状態のまま保存したのであれば「はい」を，そうでなければ「いいえ」を押下して下さい．


図 3.7.15　ウインドウ設定の保存

なお，3.7 章で作成したファイルの例として，著者の作成したファイル群をダウンロードすることができます．ホームページにてダウンロードできる「ユーザマニュアル関連情報.zip」内の「3.7 章」というディレクトリをご覧下さい．

### ３．８　来訪者受付システムのコンパイルなど（C）

いままで修正した来訪者受付システムの移動機能に関する RTC 群のコンパイルを行います.

【RHBase】

```
$cd /opt/RH/RHBase/src/comp
$sh mclean
$sh mk
$sh minst
```

【RemotePC】

```
$cd /opt/RH/RemotePC/src/comp
$sh mclean
$sh mk
$sh minst
```

これでエラーが起きなければ，来訪者受付システムの準備は終了となります.

### ３．９　便利なスクリプトファイルの作成（A）

ダウンロードした RTC 群や修正した RTC 群を実行するためにそのたび各ディレクトリを巡るのは大変面倒です. そこで, 本学習環境を実行するためのスクリプトファイル「all.sh」を作成します.

まず，来訪者受付システムの移動機能に関する RTC 群を実行するためのスクリプトファイル「start1.sh」を workspace 内に作成します．以下のソースコードをコピーするなどして，半角スペースの位置やなどに注意しながら作成して下さい.

「start1.sh」

```
-----------------------------------------------------------------
#! /bin/sh
#

RHBase_RTC_DIR=/opt/RH/RHBase/bin/comp
RHBase_CONF_DIR=/opt/RH/RHBase/etc/conf
RemotePC_RTC_DIR=/opt/RH/RemotePC/bin/comp
RemotePC_CONF_DIR=/opt/RH/RemotePC/etc/conf

exec gnome-terminal ¥
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
```

```
LocalizeCenter -e "./LocalizeCenterComp -f
${RHBase_CONF_DIR}/LocalizeCenter.conf" ¥
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
Navigation -e "./NavigationComp -f ${RHBase_CONF_DIR}/Navigation.conf" ¥
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
Odometry -e "./OdometryComp -f ${RHBase_CONF_DIR}/Odometry.conf" ¥
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
PathFollower -e "./PathFollowerComp -f ${RHBase_CONF_DIR}/PathFollower.conf" ¥
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
SwitchInputRTC -e "./SwitchInputRTCComp -f
${RHBase_CONF_DIR}/SwitchInput.conf" ¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
DispPos -e "./DispPositionComp -f ${RemotePC_CONF_DIR}/DispPosition.conf" ¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
PathPlanning -e "./PathPlanningComp -f ${RemotePC_CONF_DIR}/PathPlanning.conf"
¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
PositionInput -e "./PositionInputComp -f ${RemotePC_CONF_DIR}/PositionInput.conf"
```

---

次に，自作した RTC など来訪者受付システム以外の RTC 群を実行するためのスクリプトファイル「start2.sh」を workspace 内に作成します.

「start2.sh」

---

```
#!/bin/sh

cd /usr/share/OpenHRP-3.1/sample/controller/DataConversionRTC
#./DataConversionRTCComp &
sh bridge.sh &
cd /home/(アカウント名)/workspace/SerialPortRTC
./SerialPortRTCComp &
cd ../DataIntegrationRTC
./DataIntegrationRTCComp
```

---

次に，経路をわかりやすくするために 3.6.3 章で保存した route_map.png を RTC 群の実

行時に起動できるようにスクリプトファイル「route_map.png.sh」を workspace 内に作成します.

「route_map.png.sh」

```
----------------------------------------------------------------
#!/bin/sh

cd /opt/RH/RemotePC/bin/comp
eog route_map.png
----------------------------------------------------------------
```

また，これらを一括で起動するスクリプトファイル「all.sh」を workspace 内に作成します.

「all.sh」

```
----------------------------------------------------------------
#!/bin/sh

cd /home/(アカウント名)/workspace
sudo sh route_map.png.sh &
sudo sh start1.sh &
sudo sh start2.sh
----------------------------------------------------------------
```

また，経路計画に関連する RTC，現在位置表示に関連する RTC の動作確認に用いるために，関連した RTC のみ起動するスクリプトファイル「test2.sh」を workspace 内に作成します.

「test2.sh」

```
----------------------------------------------------------------
#! /bin/sh
#

RHBase_RTC_DIR=/opt/RH/RHBase/bin/comp
RHBase_CONF_DIR=/opt/RH/RHBase/etc/conf
RemotePC_RTC_DIR=/opt/RH/RemotePC/bin/comp
RemotePC_CONF_DIR=/opt/RH/RemotePC/etc/conf

exec gnome-terminal ¥
```

```
        --tab-with-profile=Default --working-directory=${RHBase_RTC_DIR} -t
LocalizeCenter -e "./LocalizeCenterComp -f
${RHBase_CONF_DIR}/LocalizeCenter.conf" ¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
DispPos -e "./DispPositionComp -f ${RemotePC_CONF_DIR}/DispPosition.conf" ¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
PathPlanning -e "./PathPlanningComp -f ${RemotePC_CONF_DIR}/PathPlanning.conf"
¥
        --tab-with-profile=Default --working-directory=${RemotePC_RTC_DIR} -t
PositionInput -e "./PositionInputComp -f ${RemotePC_CONF_DIR}/PositionInput.conf"
```

---

最後に，test2.sh と route_map.png.sh を同時に起動するためのスクリプトファイル「test.sh」を workspace 内に作成します.

「test.sh」

---

```
#!/bin/sh

cd /home/(アカウント名)/workspace
sudo sh route_map.png.sh &
sudo sh test2.sh
```

---

【注意点①】

～～.sh の作成方法は，3.4.1 章 SerialPortRTC の作成途中でも触れたように，

```
$cd /home/(アカウント名)/workspace
$gedit start1.sh start2.sh route_map.png.sh all.sh
```

のように，作成場所へ移動してテキストエディタで開き保存する，という方法です.

## ３．１０　経路計画に関連する RTC，現在位置表示に関連する RTC の動作確認（C）

これまでの地図・経路作成，経路データや RTC の修正に間違いがないかどうか，実際に RTC を起動して確認します．

### ３．１０．１　ネームサーバの起動

「アプリケーション＞アクセサリ＞端末」で，端末を起動します．端末が起動したら，

$sudo killall omniNames

と入力して，ネームサーバを安定稼働させるために Linux のブートプロセスとしてすでに起動しているネームサーバを削除します．このとき，パスワードの入力が求められます．入力した文字の表示はされませんので注意して下さい．次に，

$rtm-naming 2809

と入力して，ネームサーバを起動します．

### ３．１０．２　テスト用スクリプト「test. sh」の実行

「アプリケーション＞アクセサリ＞端末」で，3.10.1 章とは別の端末を起動させ，Eclipse を実行します．

$cd /home/(アカウント名)/eclipse

$sudo sh script.sh

なお，稀に Eclipse は強制終了することがあります．その際は，上記 2 行のコマンドで再度，Eclipse を起動させて下さい．

次に，Eclipse のメニューにて，「ウインドウ＞パースペクティブを開く＞その他」から RT System Editor を選択します（図 3.10.1）．

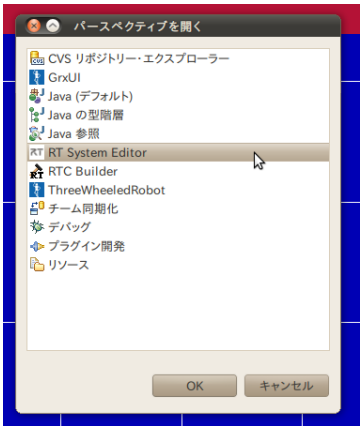


図 3.10.1　RT System Editor の選択

これで，RT System Editor が開かれます．

そして，以下のディレクトリにて 3.9 章で作成したテスト用スクリプト「test.sh」を実行します．

$cd /home/(アカウント名)/workspace

$sudo sh test.sh

実行すると，来訪者受付システムの移動機能に関する RTC 群（LocalizeCenter，DispPosition，PathPlanning，PositionInput の計 4 つの RTC）のウインドウと，3.6.3 章でスクリーンショットを撮り保存した経路地図画像のウインドウが立ち上がります（図 3.10.2）.

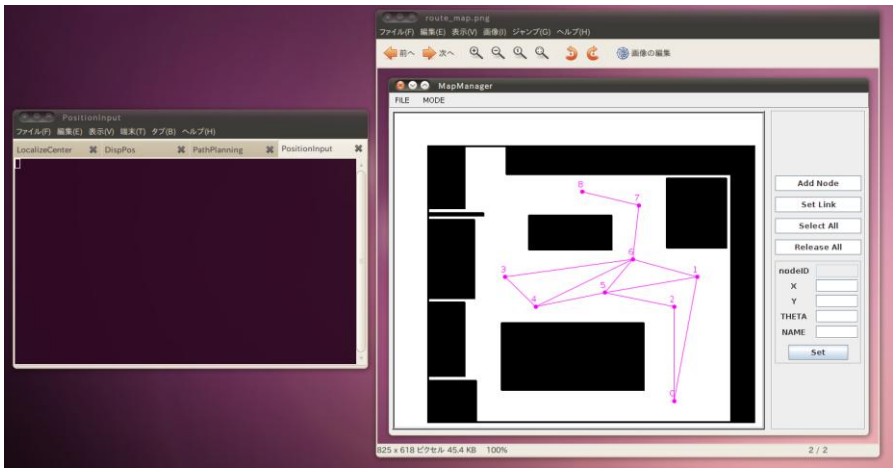


図 3.10.2　test.sh 実行後立ち上がるウインドウ

ここで，RT System Editor の左の Name Service View を見ると，各 RTC が「RTC の名称＋0」という名称で起動していることが確認できます（図 3.10.3）．ツリーが折りたたまれている場合は，図 3.10.4 のように「＋」をクリックしましょう．「＋」をクリックしても RTC が確認できない場合，図 3.10.5 に示すように更新ボタンを押下し，表示されるか確認して下さい．もし Name Service View に図 3.10.4 のような「RT localhost:2809」の表示がなければコンセントのようなボタンの「ネームサーバを追加」（図 3.10.6）で「localhost:2809」を追加して下さい（図 3.10.7）.

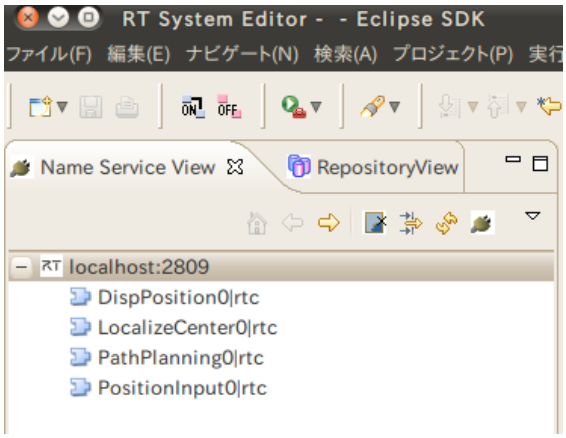


図 3.10.3　test.sh 実行後起動する RTC 群

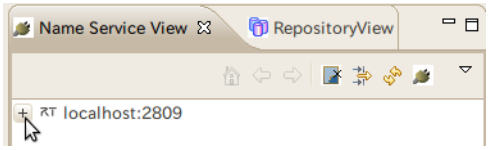


図 3.10.4　「＋」ボタンを押す

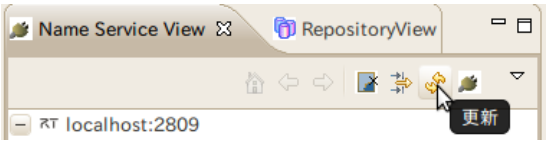


図 3.10.5　更新ボタン


図 3.10.6　ネームサーバを追加のボタン


図 3.10.7　localhost:2809 の追加

### ３．１０．３　RTC の接続，コンフィグレーションの変更

図 3.10.8 のように「Open New System Editor」のボタンをクリックして，System Diagram を開きます（図 3.10.9）.

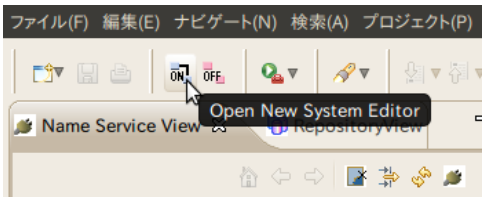


図 3.10.8　Open New System Editor

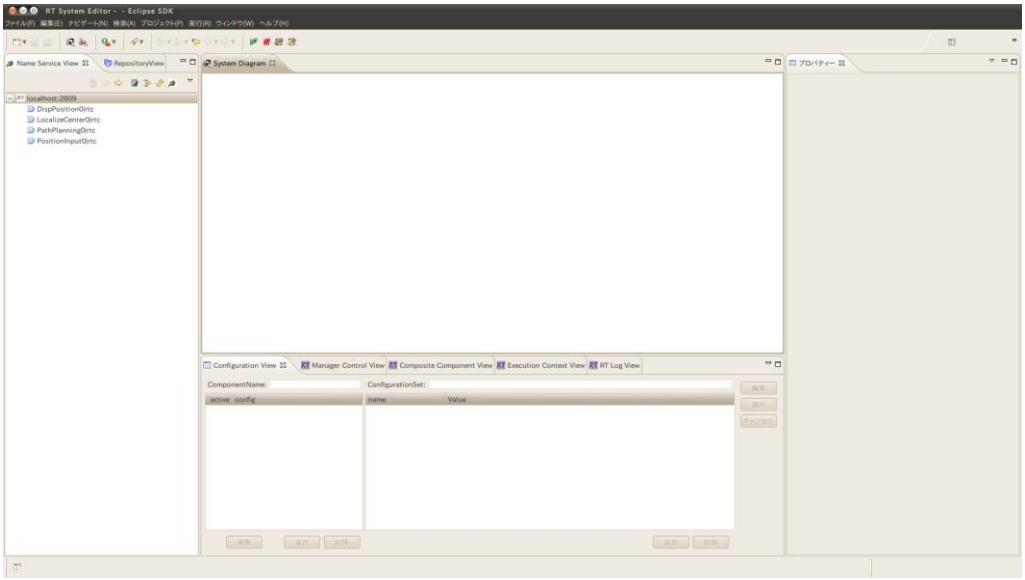

図 3.10.9　System Diagram を開いた後の RT System Editor 全体図

そして，Name Service View で起動している RTC をひとつ選択し，SystemDiagram 上にドラッグします．握り手のマークとともに RTC を選択できますので，SystemDiagram 上でマウスを離します（図 3.10.10）．すると，System Diagram 上に選択した RTC が現れますので，他の RTC も同様に System Diagram 上に生成させ，図 3.10.11 のように RTC を配置させて下さい．

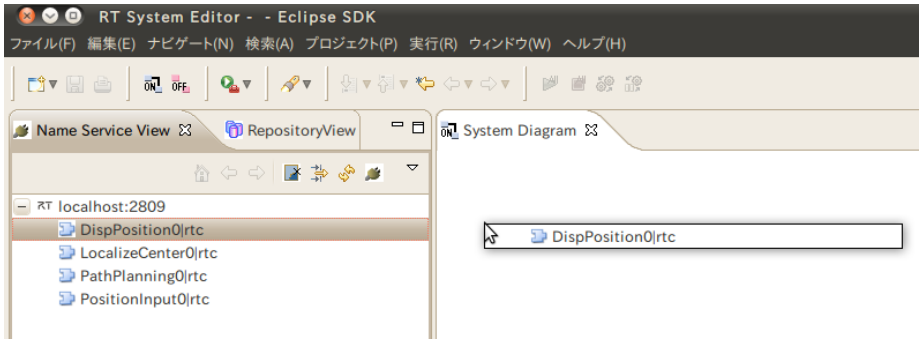


図 3.10.10　RTC を選択し System Diagram 上へ移動

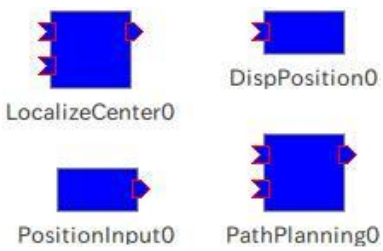


図 3.10.11　RTC の配置位置

次に，RTC の接続のために，In Port（RTC 左側にある凹部分，入力ポート）と Out Port

（RTC 右側にある凸部分，出力ポート）を結びます．まず，Out Port にマウスカーソルを合わせると，図 3.10.12 のようなマークが現れ，ドラッグすることで図 3.10.13 のような禁止のマークとともに線が出ます．その線を In Port にドラッグしていくと禁止のマークが消え，図 3.10.14 のように Out Port のときと同じマークが現れます．このときマウスのボタンを離すことで Connector Profile（図 3.10.15）というウインドウが出現しますが，これはそのまま OK を押下して下さい．すると，接続した In Port と Out Port が緑色となり，RTC 同士が接続されます．（図 3.10.16）

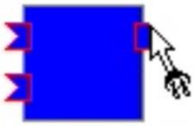

図 3.10.12　Out Port にマウスカーソルを合わせた際の様子

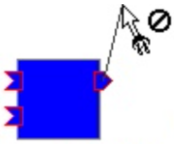

図 3.10.13　Out Port からドラッグした際の様子

図 3.10.14　In Port までドラッグした際の様子


図 3.10.15　Connector Profile

図 3.10.16　RTC の接続が完了した際の様子

これを図 3.10.16 のように全ての RTC で行って下さい.

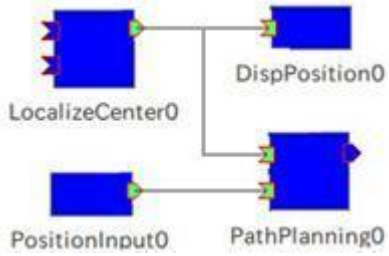


図 3.10.16　RTC の接続

次に，コンフィグレーションの設定を行います.

まず，System Diagram 上で DispPosition を選択して下さい．すると，図 3.10.17 上のように黒枠が RTC の周りに現れ，図 3.10.17 下のように Configuration View では dafult にチェックが着いている状態となっています．これは，そのままで構いません.

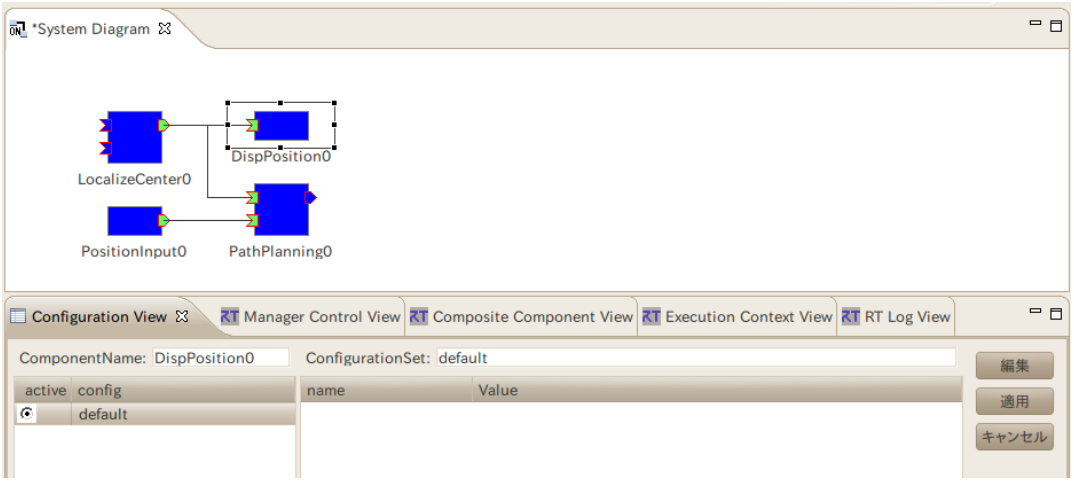


図 3.10.17　DispPosition のコンフィグレーションの設定

次に，PathPlanning を選択して下さい．すると図 3.10.18 のように Configuration View では「default」にチェックが着いている状態となっています．ここで，図 3.10.19 のように「ThreeWheeledRobot」を選択し，図 3.10.20 のように「適用」を押下して下さい.

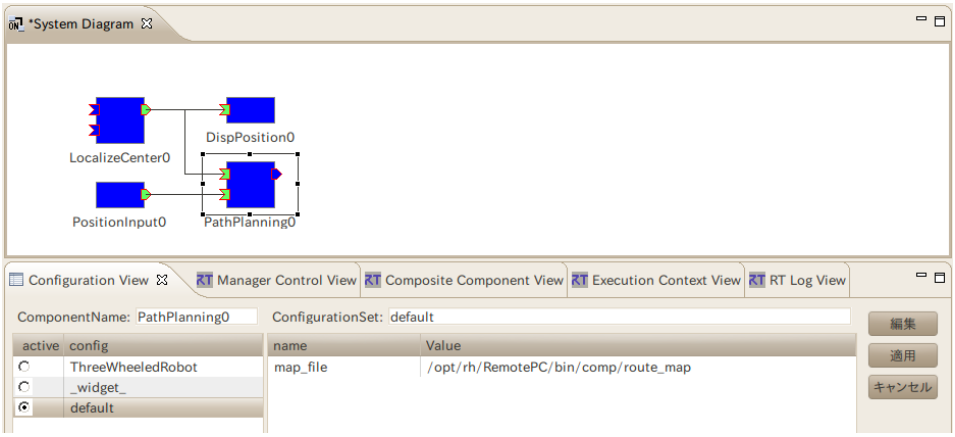


図 3.10.18　PathPlanning のコンフィグレーションの設定

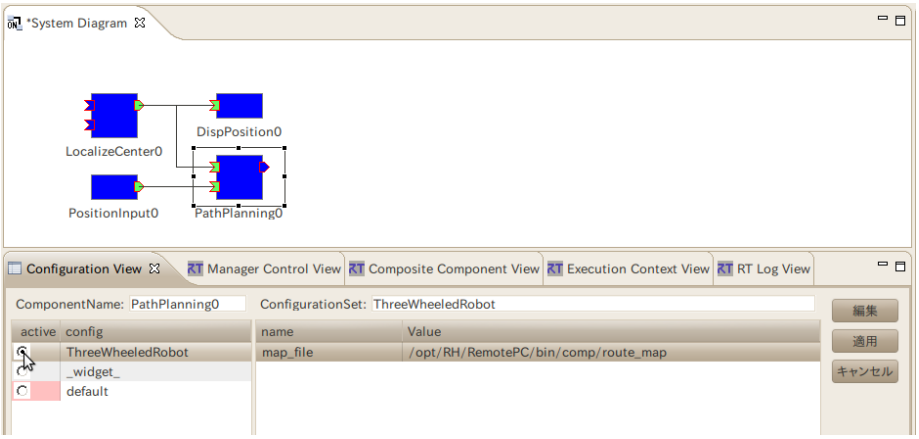


図 3.10.19　ThreeWheeledRobot の選択

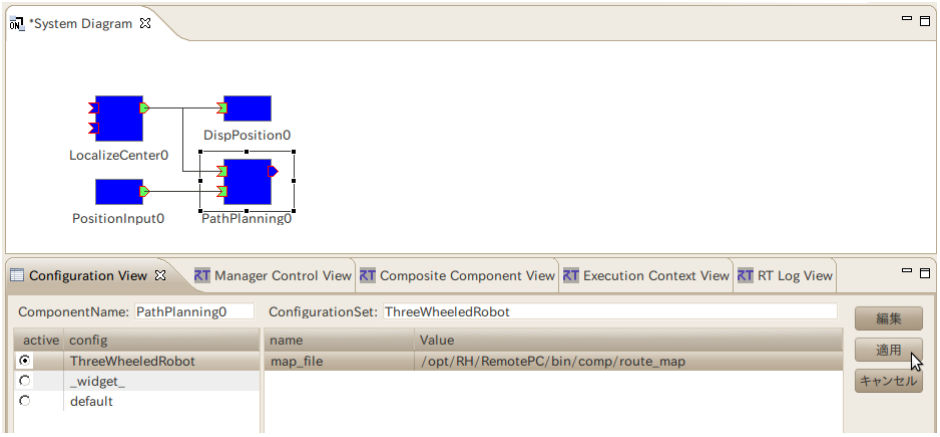


図 3.10.20　コンフィグレーションの適用

この作業を，PositionInput，LocalizeCenter でも行って下さい．PositionInput においては目標地点の座標の NodeID を，LocalizeCenter においては出発地点の座標の NodeID を選択して下さい．

### ３．１０．４　RTC 動作確認

図 3.10.21 のように All Activate のボタンをクリックし, 全ての RTC を Activate(緑色)にします.

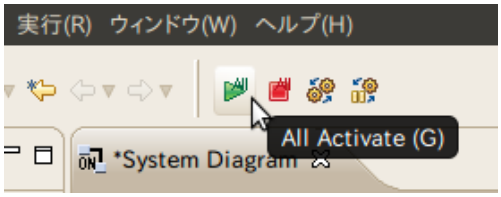


図 3.10.21　All Activate

すると，図 3.10.22 のような新しいウインドウが 2 つ現れます.

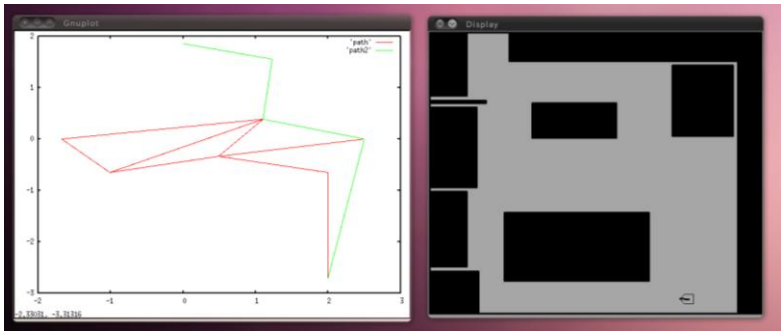

図 3.10.22　Activate 後に出現するウインドウ群の例

図 3.10.22 左のウインドウでは 3.6.4 章で修正した経路データが表示されており，出発地点から目標地点までの経路が緑色で，その他使用しない経路は赤色で表示されております．また，図 3.10.22 右のウインドウでは，作成した地図画像の上に，3.6.4 章で修正した RTC「DispPosition」によりロボットの初期位置が表示されております．これらが意図した通り表示されていることを確認し，もし表示されていない場合は修正した各章へと戻って間違いがないか確認して下さい.

正常に表示されている場合，図 3.10.23 のように All Deactivate をクリックします.

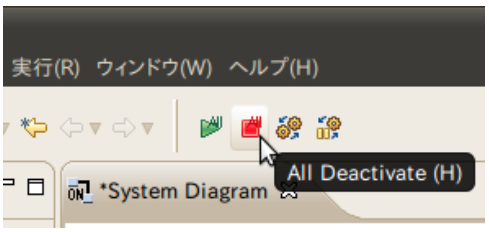


図 3.10.23　All Deactivate

これで，すべて Deactivate（青色）となり，動作確認は終了です.

## ３．１１　3 輪移動ロボットの製作（A）

### ３．１１．１　部品表

部品表はホームページにてダウンロードできる「3 輪移動ロボット･ハードウェア情報.zip」内にあります.

### ３．１１．２　電子回路図

電子回路図（図 3.11.1）はホームページにてダウンロードできる「3 輪移動ロボット・ハードウェア情報.zip」内にあります.

電子回路図は P 板.com より無償配布されている電子回路設計 CAD ソフト「CADLUS サーキット」を用いて設計を行いました.

（　プリント基板 ネット通販 P 板.com（ピーバンドットコム）: http://www.p-ban.com/　）

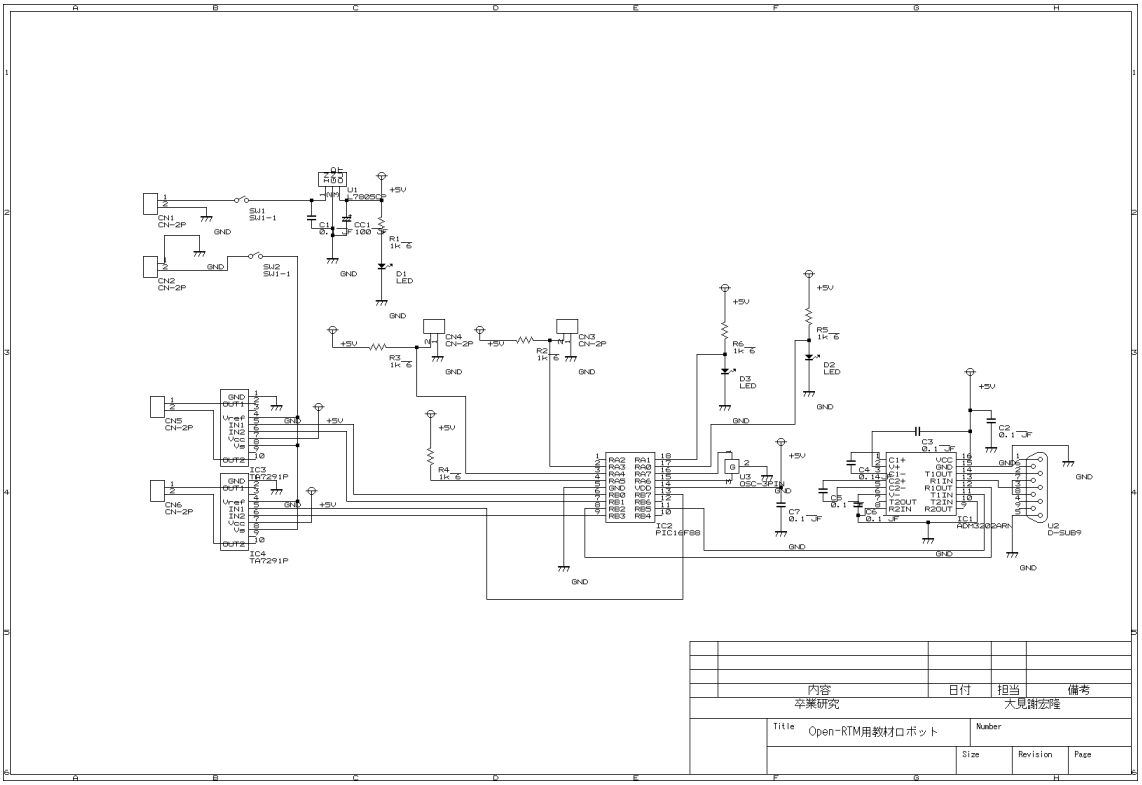


図 3.11.1　電子回路図

### ３．１１．３　ガーバデータ

ガーバデータはホームページにてダウンロードできる「3 輪移動ロボット・ハードウェア情報.zip」内にあります.

設計した電子回路図をもとに, P 板.com より無償配布されているプリント基板設計 CAD ソフト「CADLUS X」を用いて電子回路図をプリント基板として設計しました.

### ３．１１．４　PIC プログラム

PIC プログラムはホームページにてダウンロードできる「3 輪移動ロボット・ハードウェア情報.zip」内にあります.

また，PIC の開発環境を表 3.11.1 に示します.

表 3.11.1　PIC 開発環境

| | |
|---|---|
| コンパイラソフト | Mikro-C Pro for PIC (無償版) |
| 書き込みソフト | PIC kit 2 v2.61 |
| 書き込み用ハード | PIC kit2 |

### ３．１１．５　製作のための参考画像

製作の参考となる画像を図 3.11.2～図 3.11.18 に示します.

ギアボックスは A タイプを，ボールキャスターは高さ 35mm のものを製作して下さい.

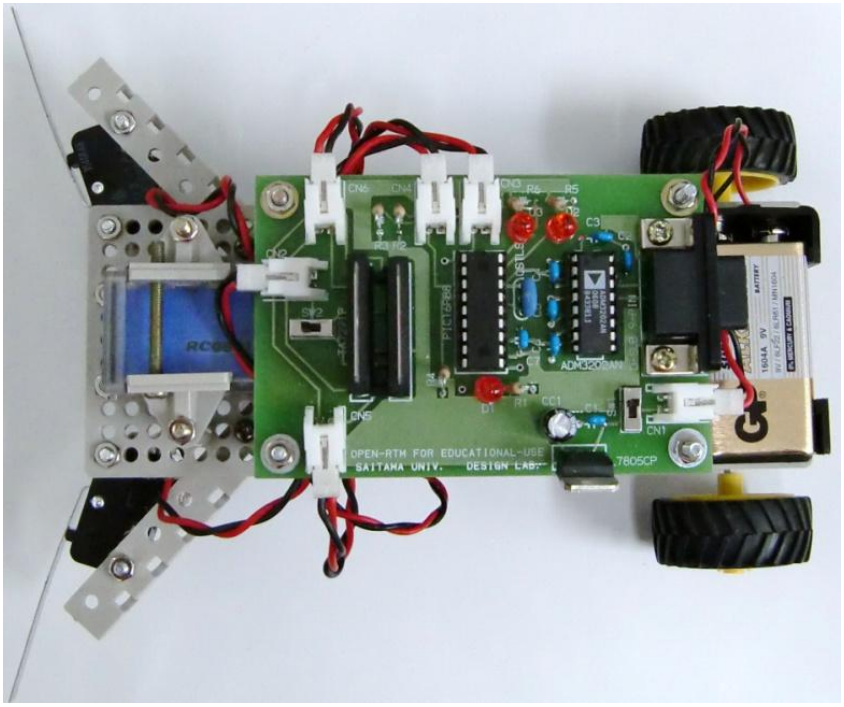


図 3.11.2　上方向からのロボット

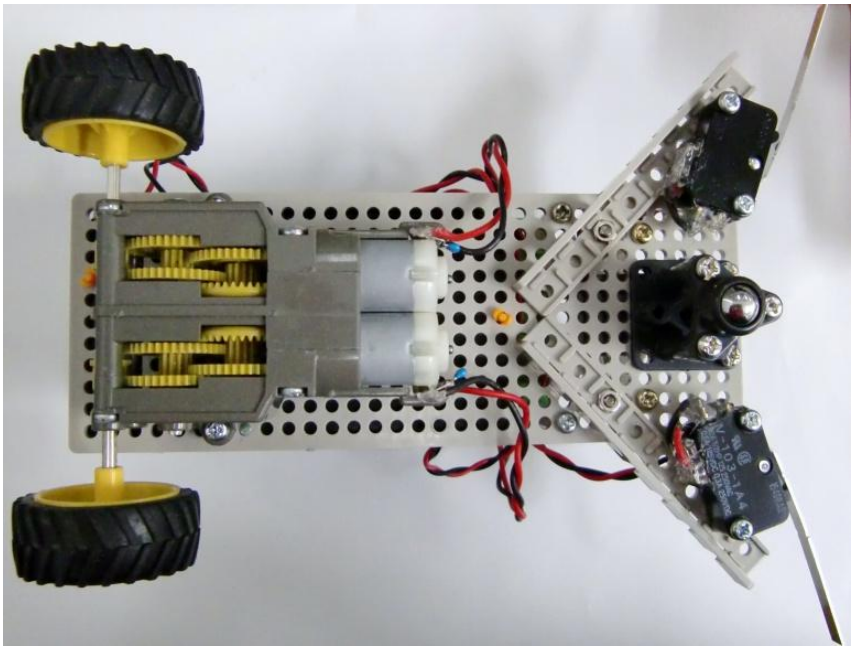

図 3.11.3　下方向からのロボット

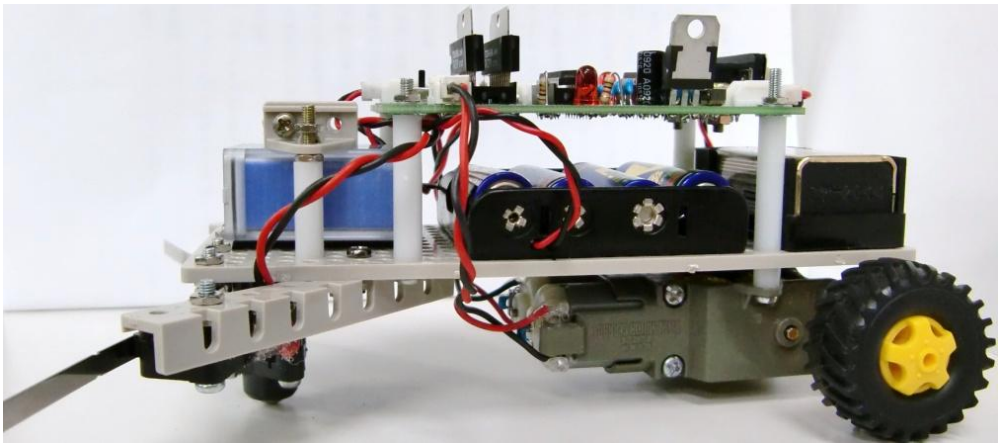

図 3.11.4　左横方向からのロボット

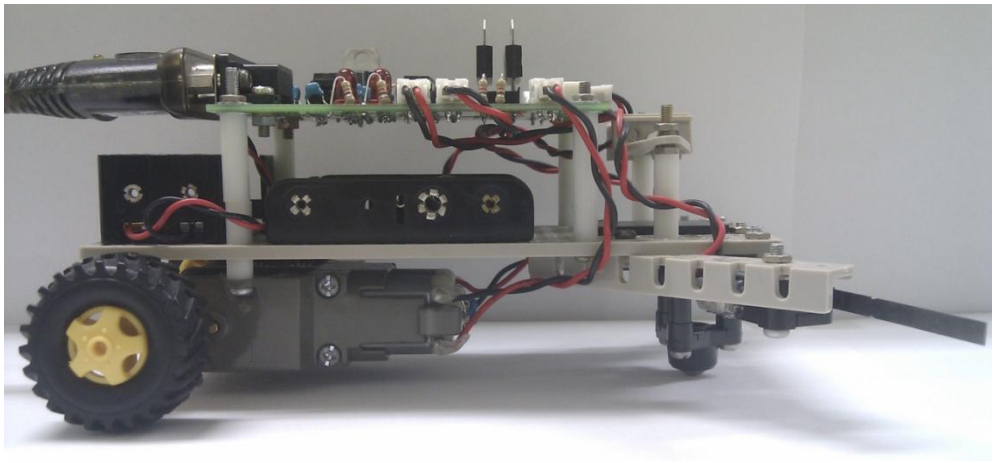

図 3.11.5　右横方向からのロボット

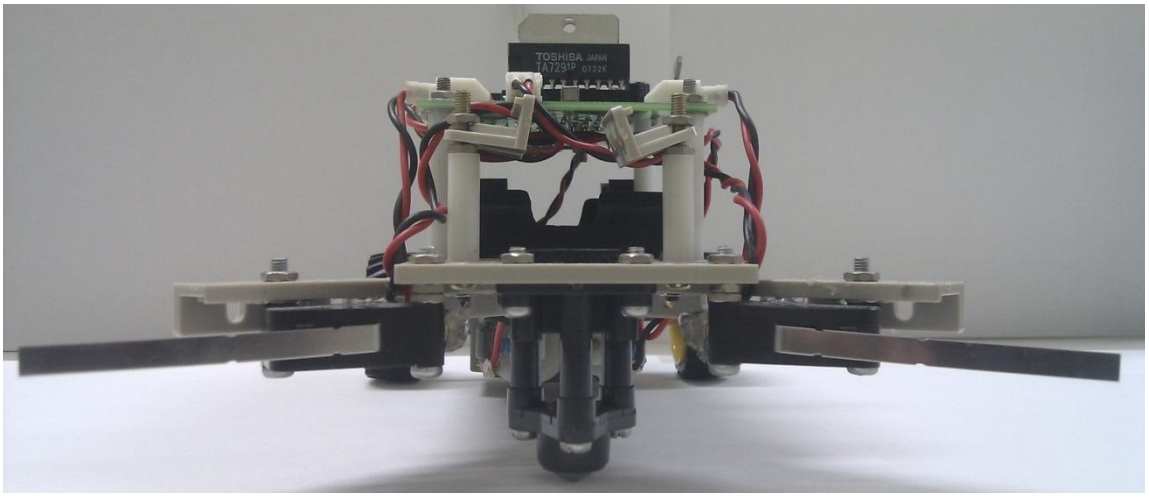


図 3.11.6　前方からのロボット

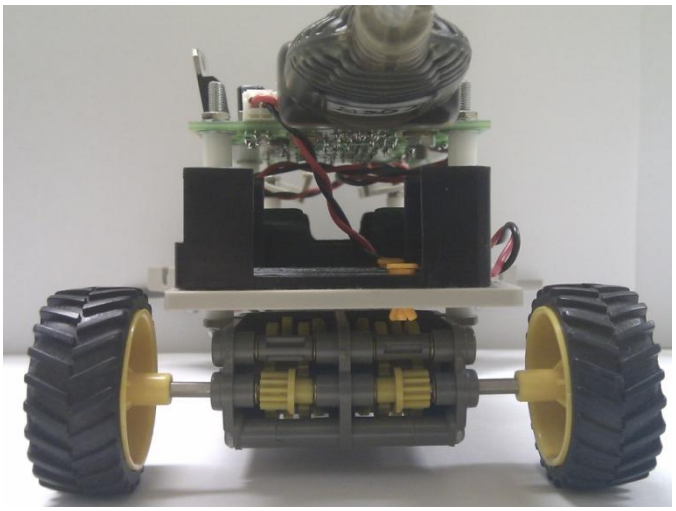

図 3.11.7　後方からのロボット

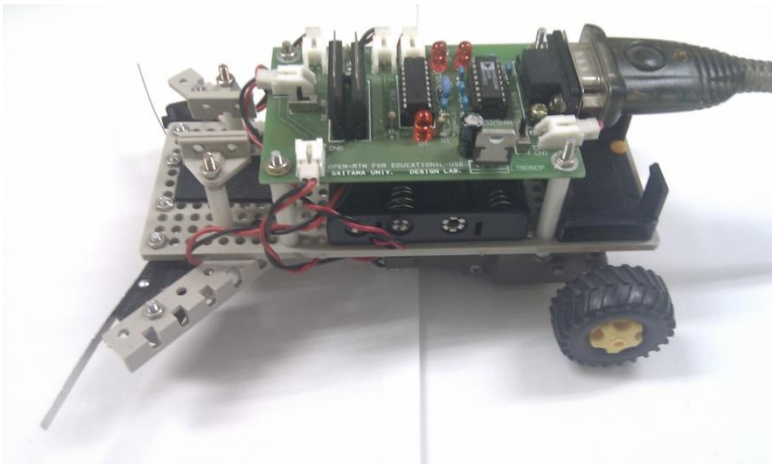

図 3.11.8　左斜め上からのロボット

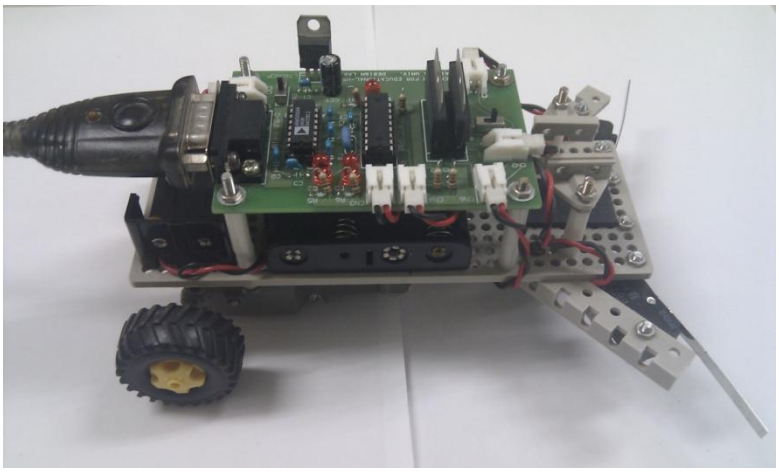

図 3.11.9　右斜め上からのロボット

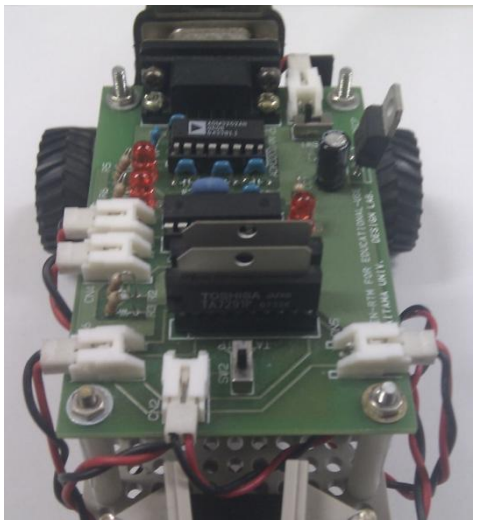

図 3.11.10　前方斜め上からのロボット

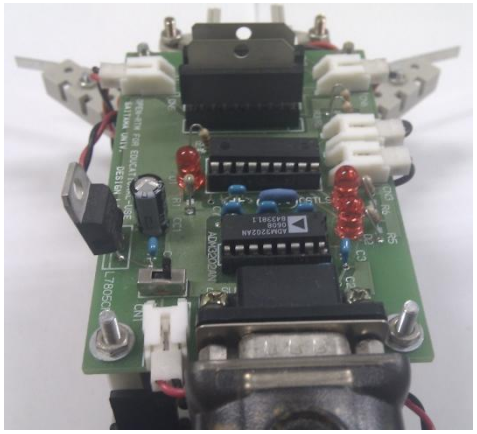

図 3.11.11　後方斜め上からのロボット

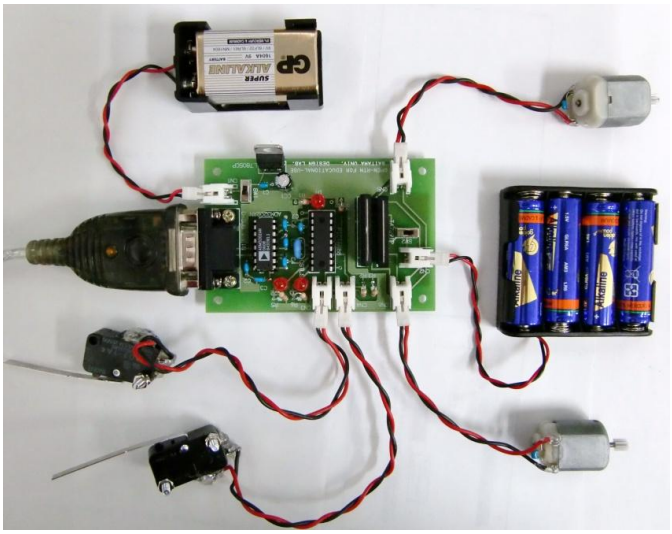

図 3.11.12　ロボットの制御回路

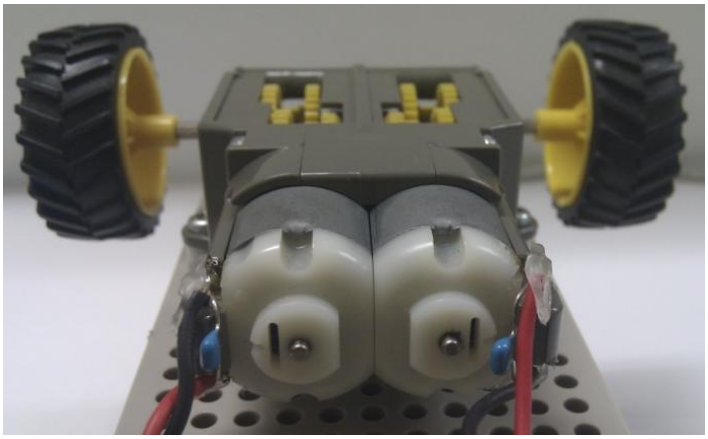

図 3.11.13　モータに用いるセラミックコンデンサ（0.1μF）

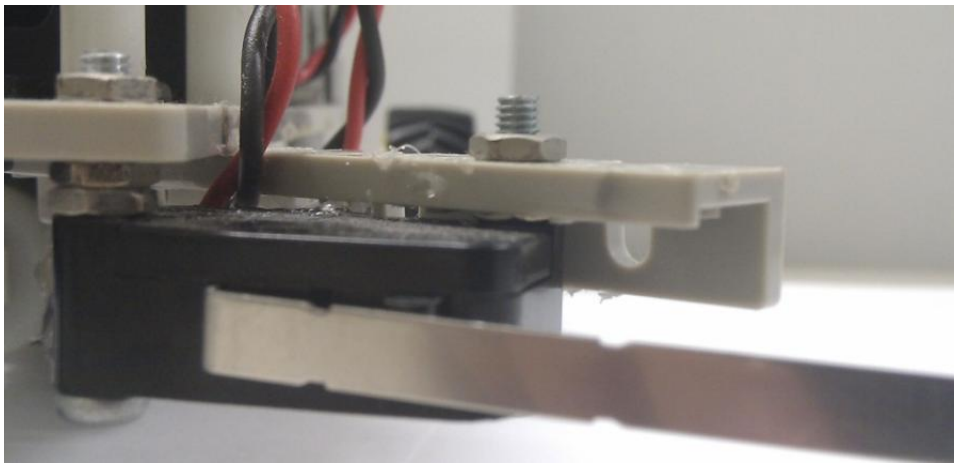

図 3.11.14　マイクロスイッチ付近のねじ止め

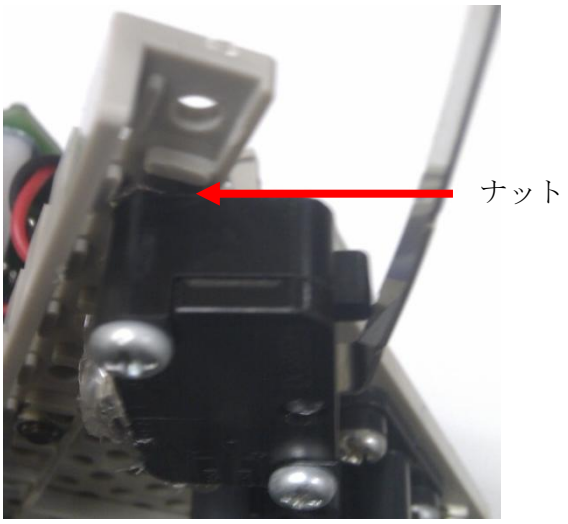


図 3.11.15　マイクロスイッチ上部のナット

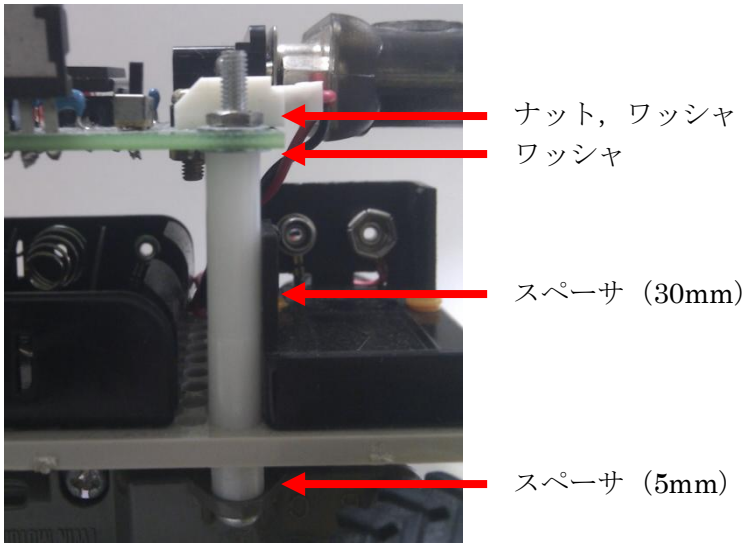


図 3.11.16　プリント基板とユニバーサルプレート，ギアボックスのねじ止め

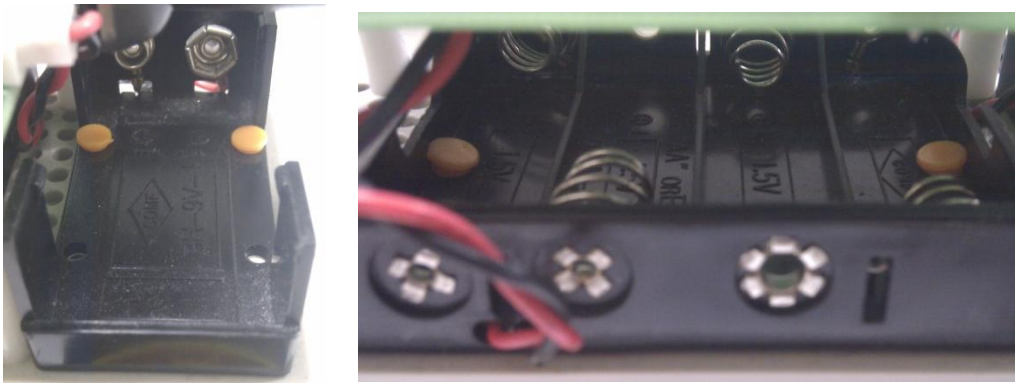

図 3.11.17　電池ボックスのユニバーサルプレートへの固定方法

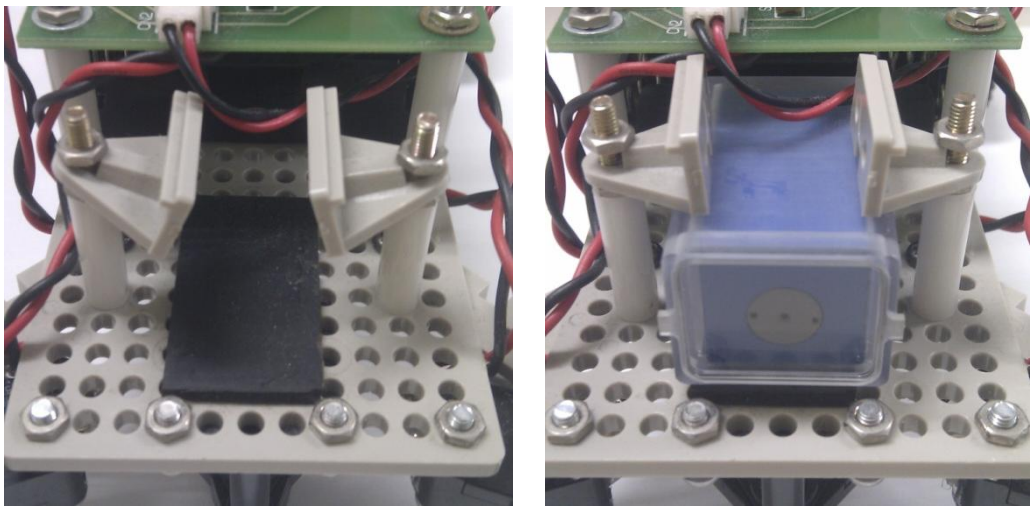

図 3.11.18　無線カメラ固定前（左図）・固定後（右図）

### ３．１１．６　製作における注意点

部品表，回路図，ガーバデータ，PIC プログラムがダウンロードできますが，これらはあくまで一例となっています．本 3 輪移動ロボットは，機能を拡充したりより安価にしたりなど自由に改良できることも利点となっているため，3.11 章でダウンロードできるデータは自由に変更して頂いて構いません．

### ３．１２　3 輪移動ロボットの動作確認（A）

ロボットが正常に動作するかの確認を行います．

ロボットを PC に接続します．まず，図 3.12.1 のように 3 輪移動ロボットの後方に USB シリアル変換ケーブルを接続して下さい．そして，図 3.12.2 に示すような USB2.0 延長ケーブルを用意し（図 3.12.2 は長さ 5m のもの），図 3.12.3 のように USB シリアル変換ケーブルと接続して下さい．なお，USB2.0 延長ケーブルは両端が図 3.12.4，図 3.12.5 のような端子となっています．図 3.12.4 の端子を USB シリアル変換ケーブルに接続して下さい．また，図 3.12.5 の端子は，図 3.12.6 のように PC の USB ポートへ接続して下さい．PC から 3 輪移動ロボットまでの接続の全体図は，図 3.12.7 のようになります．

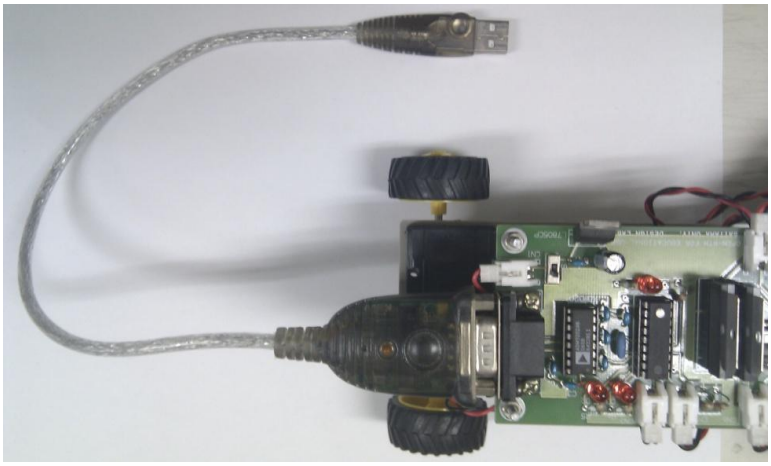

図 3.12.1　3 輪移動ロボットと USB シリアル変換ケーブルとの接続

図 3.12.2　長さ 5m の USB2.0 延長ケーブル

図 3.12.3　USB2.0 延長ケーブルと USB シリアル変換ケーブルとの接続

図 3.12.4　USB2.0 延長ケーブルの USB シリアル変換ケーブルに接続する端子

図 3.12.5　USB2.0 延長ケーブルの PC へ接続する端子

図 3.12.6　USB2.0 延長ケーブルと PC の USB ポートとの接続

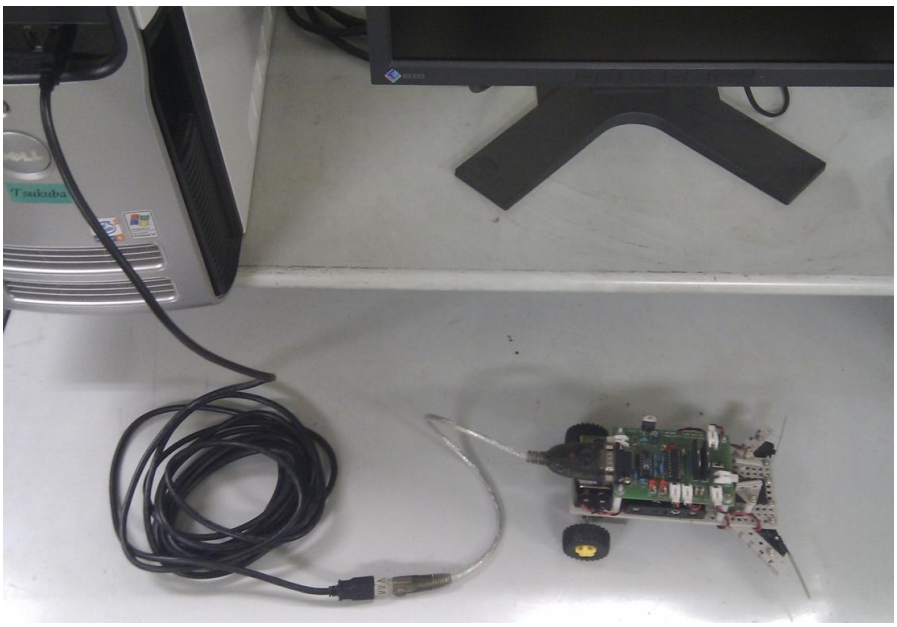

図 3.12.7　3 輪移動ロボットから PC までの接続の全体図

次に，ロボットの電源スイッチを入れます．図 3.12.8 の左のスイッチがモータ用電源スイッチ，右のスイッチが制御回路用電源スイッチとなっております．それぞれ ON にして下さい．

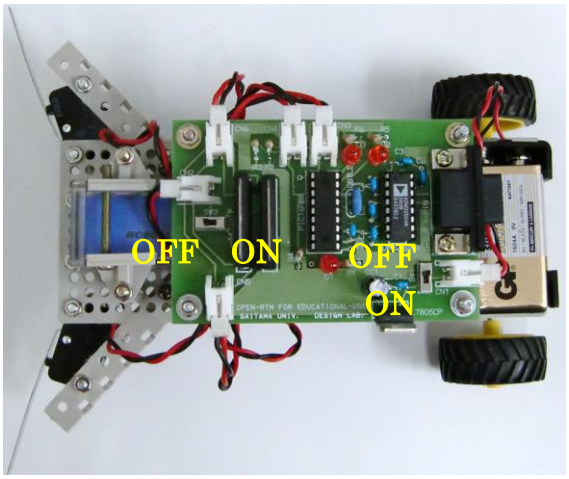


図 3.12.8　3 輪移動ロボットの電源位置

3 輪移動ロボットの動作確認を行います．図 3.12.9 のように「アプリケーション＞アクセサリ＞Serial port terminal」で，GtkTerm を起動します．次に，GtkTerm から「Configuration＞Port」を選択し（図 3.12.10），図 3.12.11 や表 3.12.1 のように Configuration を設定し，OK を押下します．また，図 3.12.10 において，「Local echo」にチェックを入れて下さい．なお接続が USB の場合，「Port」では「/dev/ttyS0」から「/dev/ttyUSB0」への変更を忘れないで下さい．


図 3.12.9　アプリケーション＞アクセサリ＞Serial port terminal


図 3.12.10　Configuration>Port


図 3.12.11　GtkTerm の Configuration 設定画面

表 3.12.1　GtkTerm の Configuration 設定

| | |
|---|---|
| Port | 適切な接続先（例：/dev/ttyUSB0） |
| Speed | 9600 |
| Parity | none |
| Bits | 8 |
| Stopbits | 1 |
| Flow control | none |

表 3.12.2，表 3.12.3 に従ってロボットに指令値を送り，動作確認を行います．指令値は，「表 3.12.2 の値」＋「表 3.12.3 の値」＋「o」＋「k」という仕様となっております．例えば 1aok は最高速度での前進，5bok は最低速度での後退などと定義されております．なお，「1aok」は「いち・えー・おー・けー」であり，指令値はすべて「1～5 までの数字」＋「a～k までのアルファベット」＋「o（おー，ゼロや O（o の大文字）ではない）」＋「k（けー）」という仕様となっております．

また，GtkTerm にて入力を行う際，指令値以外の入力，例えば Enter キーの押下など，は行わないで下さい．ロボットが入力を受け付けなくなる恐れがあります．つまり，GtkTerm にて指令値を変更したい場合は，「1aok1bok」のように，連続して入力して下さい．この場合は，1aok の入力終了時点で最高速度での前進を行い，1bok の入力終了時点で最低速度での後退を行うようになります．

もし入力を受け付けなくなった場合は，一度ロボットの電源を OFF にし，再度最初から手順通り行って下さい．

表 3.12.2　速度を決定する文字データ

| 速度 | 受け取るデータ |
|---|---|
| 最高速度 | 1 |
| 高速 | 2 |
| 通常速度 | 3 |
| 低速 | 4 |
| 最低速度 | 5 |

表 3.12.3　動作を決定する文字データ

| 動作 | 受け取るデータ |
|---|---|
| 前進 | a |
| 後退 | b |
| 右折 | c |
| 急な右折 | d |
| 左折 | e |
| 急な左折 | f |
| 右後退 | g |
| 急な右後退 | h |
| 左後退 | i |
| 急な左後退 | j |
| 停止 | k |

表 3.12.2，表 3.12.3 に従って適宜動作確認を行っても動作しなかった場合は，簡単なようなら原因を調査し，難しいようならロボットの交換を行って下さい．

正常に動作した場合，ロボットの動作確認は終了です．電池の節約のため，ロボットの電源を両方とも OFF にして下さい．

# ４　RT ミドルウェア学習環境を使う

## ４．１　3 輪移動ロボットの準備

ロボットを PC に接続します．まず，図 4.1.1 のように 3 輪移動ロボットの後方に USB シリアル変換ケーブルを接続して下さい．そして，図 4.1.2 に示すような USB2.0 延長ケーブルを用意し（図 4.1.2 は 5m のもの），図 4.1.3 のように USB シリアル変換ケーブルと接続して下さい．なお，USB2.0 延長ケーブルは両端が図 4.1.4，図 4.1.5 のような端子となっています. 図 4.1.4 の端子を USB シリアル変換ケーブルに接続して下さい. また, 図 4.1.5 の端子は，図 4.1.6 のように PC の USB ポートへ接続して下さい．PC から 3 輪移動ロボットまでの接続の全体図は，図 4.1.7 のようになります.

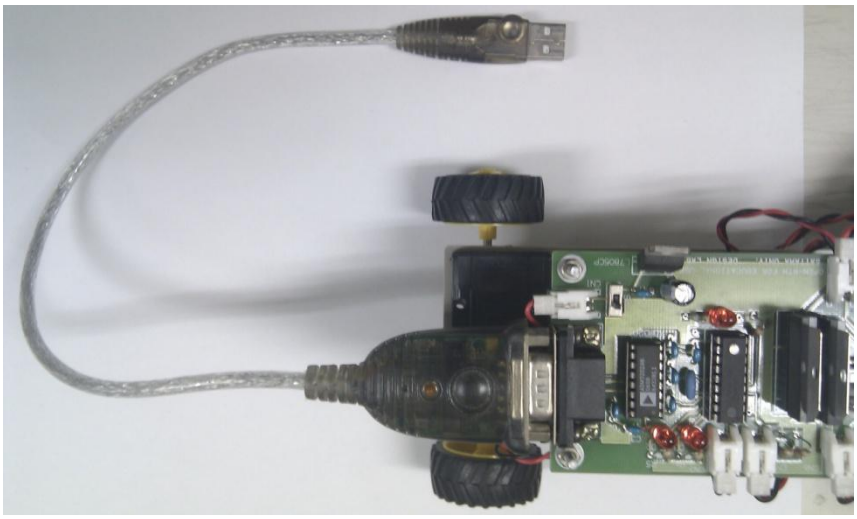

図 4.1.1　3 輪移動ロボットと USB シリアル変換ケーブルとの接続

図 4.1.2　5m の USB2.0 延長ケーブル

図 4.1.3　USB2.0 延長ケーブルと USB シリアル変換ケーブルとの接続

図 4.1.4　USB2.0 延長ケーブルの USB シリアル変換ケーブルに接続する端子

図 4.1.5　USB2.0 延長ケーブルの PC へ接続する端子

図 4.1.6　USB2.0 延長ケーブルと PC の USB ポートとの接続

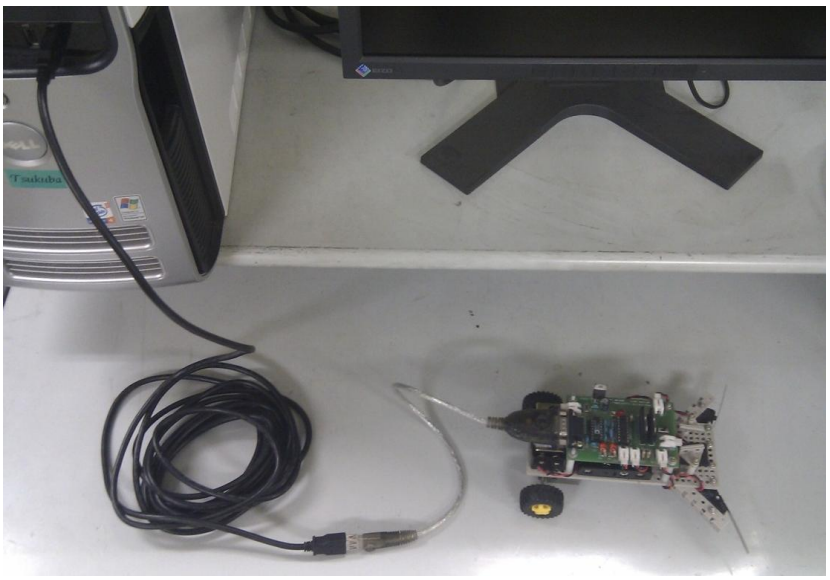

図 4.1.7　3 輪移動ロボットから PC までの接続の全体図

次に，ロボットの電源を入れます．図 4.1.8 に示した電源位置の通り，両方 ON にして下さい．

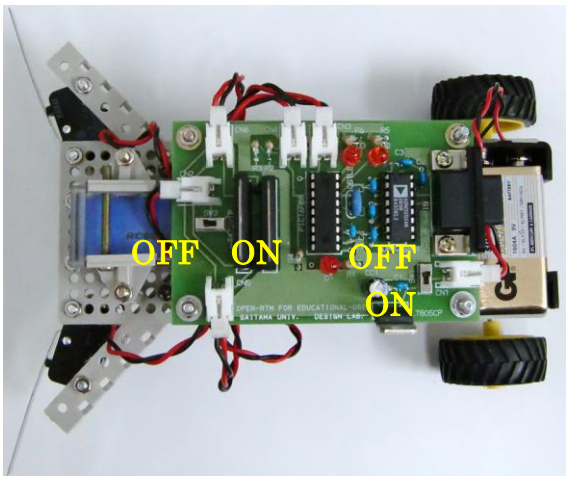


図 4.1.8　3 輪移動ロボットの電源位置

### ４．２　ネームサーバの起動

「アプリケーション＞アクセサリ＞端末」で，端末を起動します．端末が起動したら，

$sudo killall omniNames

と入力して，ネームサーバを安定稼働させるために Linux のブートプロセスとしてすでに起動しているネームサーバを削除します．次に，

$rtm-naming 2809

と入力して，ネームサーバを起動します．

## ４．３　OpenHRP3 のプロジェクト読み込み

まず，「アプリケーション＞アクセサリ＞端末」で，4.2 章とは別の端末を起動します．端末が起動したら，以下のように Eclipse を起動します．

$cd /home/(アカウント名)/eclipse

$sudo sh script.sh

次に，Eclipse のメニューにて「ウインドウ＞パースペクティブを開く＞その他」(図 4.3.1) で，GrxUI を選択し（図 4.3.2）実行します．

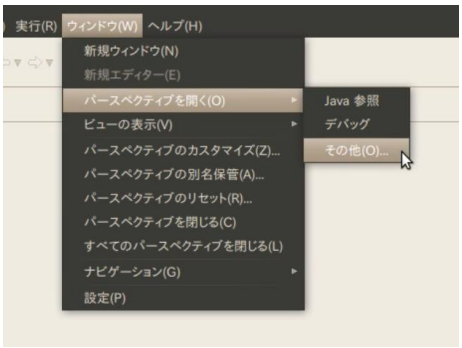


図 4.3.1　パースペクティブを開く

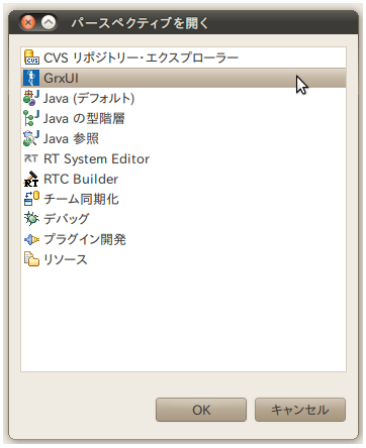


図 4.3.2　GrxUI の選択

これで GrxUI が開くはずですが，アイテムや 3D ビューなどが表示されないことがあります（図 4.3.3）．そこで，図 4.3.4 のように「ウインドウ＞新規ウインドウ」で新規ウインドウを開きます．新規ウインドウが開かれたら前のウインドウを終了することで，GrxUI が正常に実行可能となります．（図 4.3.5）

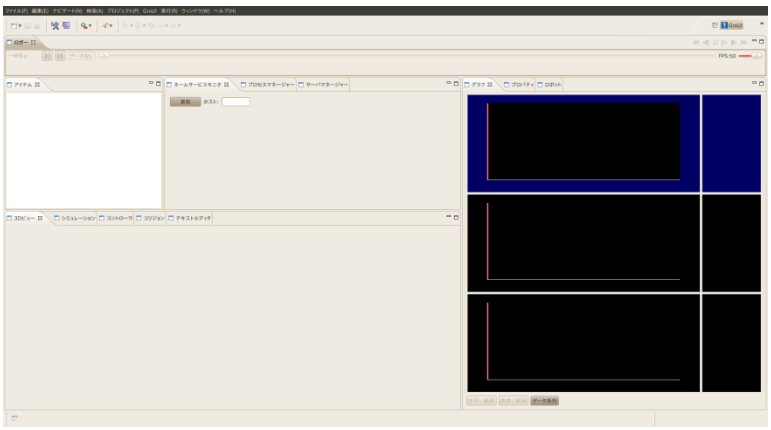

図 4.3.3　GrxUI でアイテムなどが表示されない場合


図 4.3.4　新規ウインドウを開く

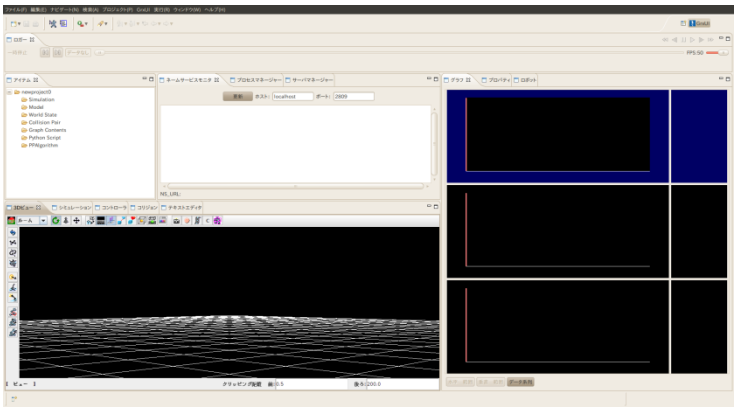

図 4.3.5　GrxUI でアイテムなどが表示されている場合

そして，「newproject0 の上で右クリック＞プロジェクトの読み込み」（図 4.3.6）で，3.7.2 章で保存したシミュレーション環境「ThreeWheeledRobot.xml」を選び（図 4.3.7），3D ビューの左上にある「ルーム」を変更し「上」にします（図 4.3.8）．これは，ロボットモデル

の移動経路を見やすくするためであり，状況に応じて 3D ビューの視点は変更してかまいません．その後，その右にある「ズーム」のボタンをクリックし（図 4.3.9），画面内にシミュレーション環境がすべて入るようにします．

VRML の描画が正常でないとき（図 4.3.10）は，右クリックをしながら左右にドラッグすることで見えるようになります（図 4.3.11）．


図 4.3.6　プロジェクトの読み込み

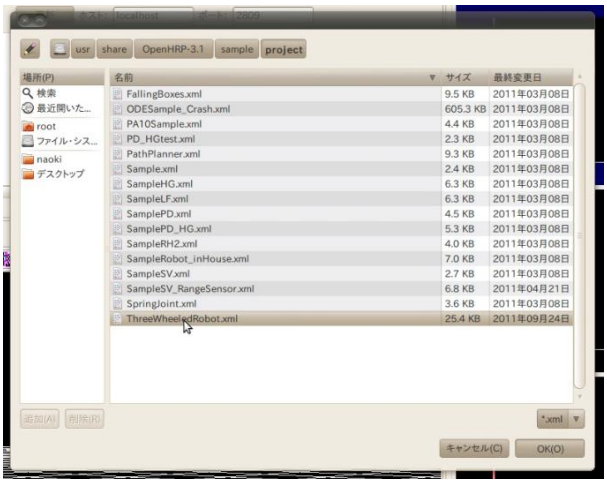


図 4.3.7　ThreeWheeledRobot.xml の選択


図 4.3.8　「ルーム」から「上」へ


図 4.3.9　「ズーム」をクリック

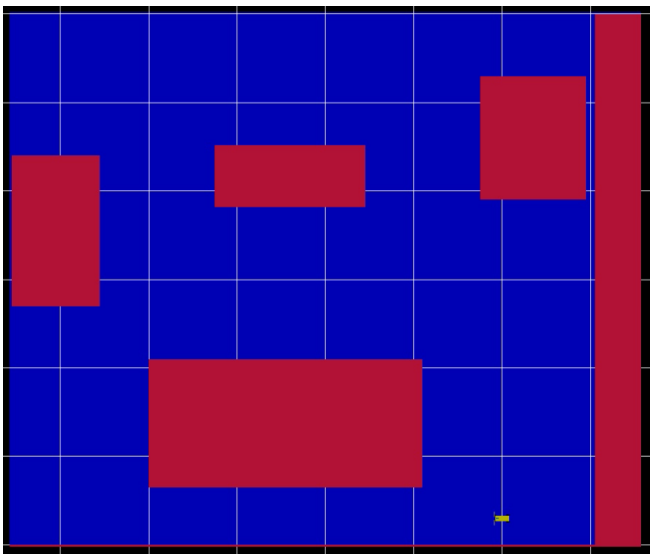

図 4.3.10　正常でない描画例

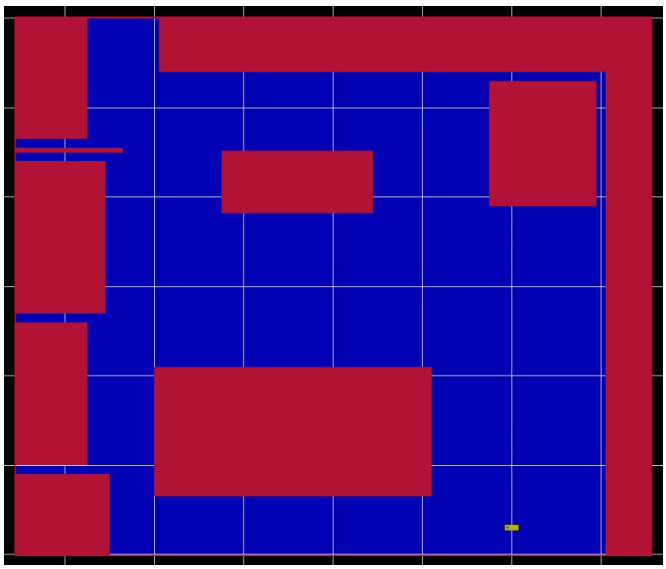

図 4.3.11　正常な描画例

これで，プロジェクトの読み込みが終了しました.

## ４．４　RTC 一括起動スクリプト「all.sh」の実行

端末を開き，以下のように all.sh を保存している場所まで移動し，all.sh を実行します．

$cd /home/(アカウント名)/workspace

$sudo sh all.sh

実行すると，来訪者受付システムの移動機能に関する RTC 群のウインドウ（図 4.4.1 上）と，経路地図のウインドウ（図 4.4.1 右）が立ち上がります．また，既に開いていた端末（図 4.4.1 左）には「ready」と表示されます．

このとき，来訪者受付システムの移動機能に関する RTC 群のウインドウのタブひとつひとつ（RTC ひとつひとつのタブ）を見て，「Initialized」のように表示されている文章を確認することで，RTC で何が起きているか知ることができます．RTC の実行中に知りたいことがあった場合は，該当する RTC のウインドウを確認すると良いでしょう．

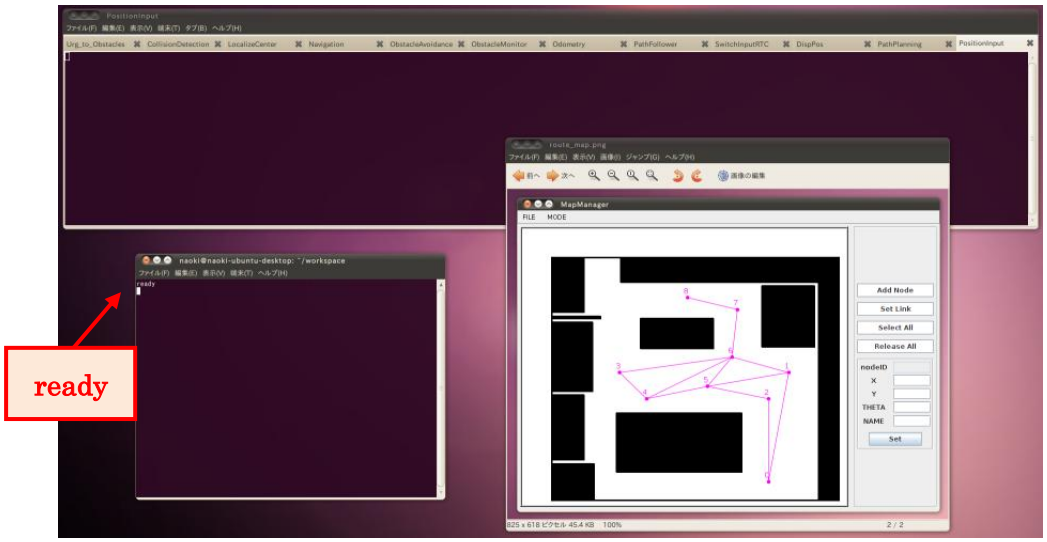


図 4.4.1　all.sh 実行後のウインドウ例

次に，Eclipse のメニューより，「ウインドウ＞パースペクティブを開く＞その他」から RT System Editor を選択します．3.10.2 章でも述べた通り，RT System Editor の左の Name Service View を見ると，図 4.4.2 のように各 RTC が「RTC の名称＋0」という名称で起動していることが確認できます．なお，図 4.4.2 の「？」のアイコンは RTC ではありません．表示されない場合は，3.10.2 章の方法でネームサーバの追加や更新などを行って下さい．

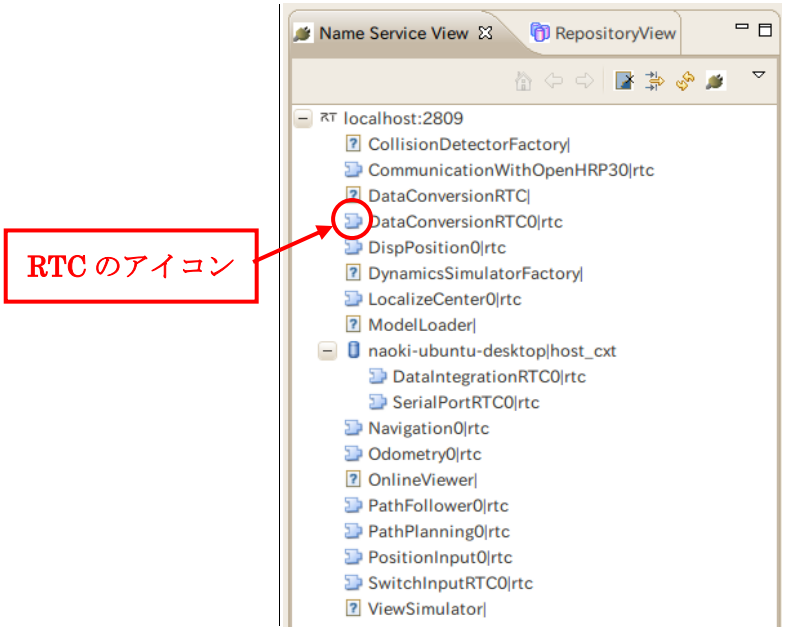


図 4.4.2　現在起動している RTC

ここで先ほど GrxUI で開いたプロジェクト「ThreeWheeledRobot.xml」に戻ります.「3D ビュー」などのタブに並んでいる,「コントローラ」の設定を行います.

ロボット名 - MODEL_CAR を選択し，下の「編集」をクリックします．そして，コントローラ名を「DataConversionRTC」（DataConversionRTC0 ではない），実行ディレクトリを「/usr/share/OpenHRP-3.1/sample/project」のままとし，実行コマンドは空欄，コントロール時間[秒]を 0.003 に設定して下さい．そして左下の「了解」をクリックし，3D ビューに戻ります．ここで，図 4.4.3 のようにプロジェクトの保存を行って下さい．

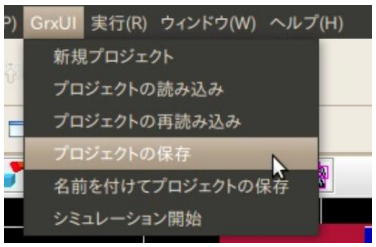


図 4.4.3　プロジェクトの保存

ここで再度，RT System Editor に戻ります.

## ４．５　RTC の接続

3.10.3 章と同様に，図 4.5.1 のように「Open New System Editor」のボタンをクリックして，System Diagram を開きます.

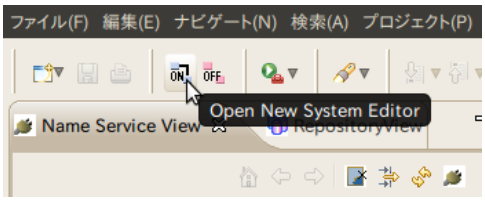


図 4.5.1　Open New System Editor

次に，Name Service View で起動している RTC をひとつ選択し，図 4.5.2 のように SystemDiagram 上にドラッグします. 握り手のマークとともに RTC を選択できますので，SystemDiagram 上でマウスを離します．すると，System Diagram 上に選択した RTC が現れますので，他の RTC も同様に System Diagram 上に生成させ，図 4.5.3 のように RTC を配置させて下さい.

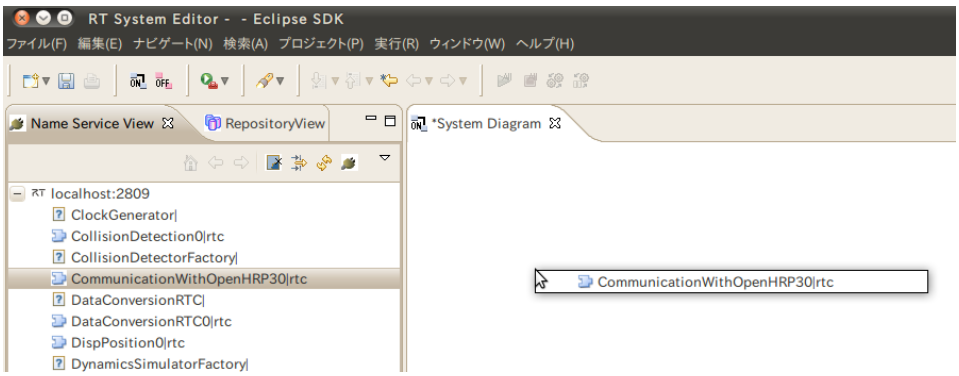


図 4.5.2　RTC を選択し System Diagram 上へ移動

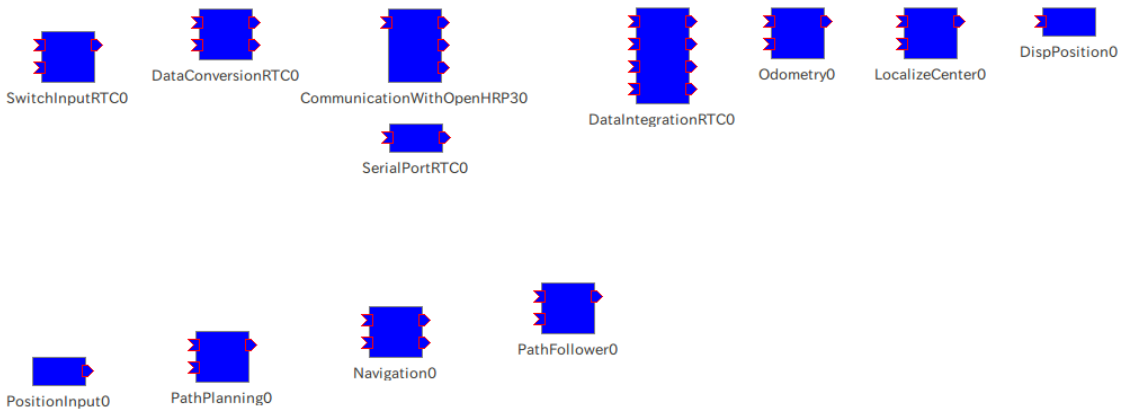


図 4.5.3　RTC の配置位置

次に，RTC の接続を図 4.5.4 のように全ての RTC で行って下さい.

図 4.5.4 では重なって表示されているため，接続が判断しにくいポートのみ解説します．まず，LocalizeCenter0 からの出力は，DispPosition0，Odometry0，PathPlanning0，Navigation0，PathFollower0 に接続して下さい．CommunicationWithOpenHRP30 の上の Out Port は，DataIntegrationRTC0 の上の In Port と，DataConversionRTC0 の上の In Port に接続して下さい．

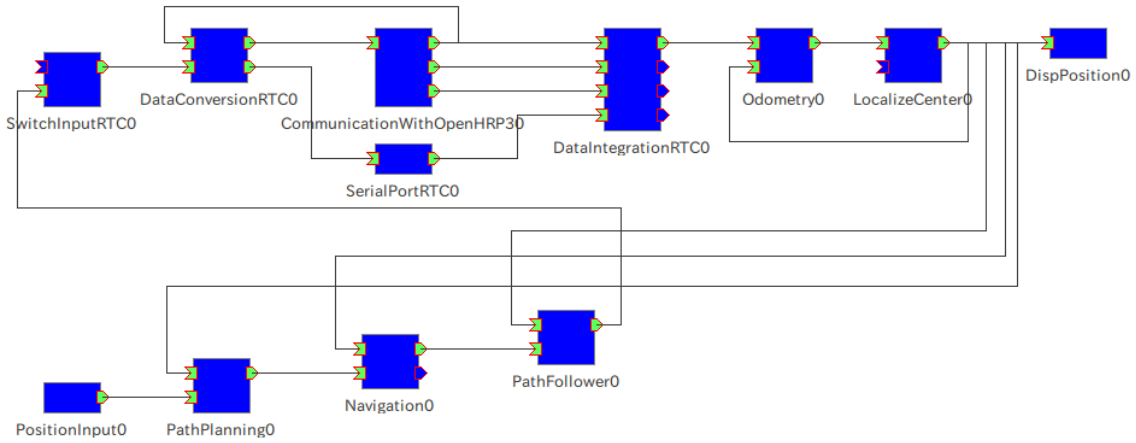


図 4.5.4　RTC の接続

ここで，System Diagram 上の適当な位置で右クリックして，「Save As」を選択します．すると，図 4.5.5 のようにウインドウが表示されます．

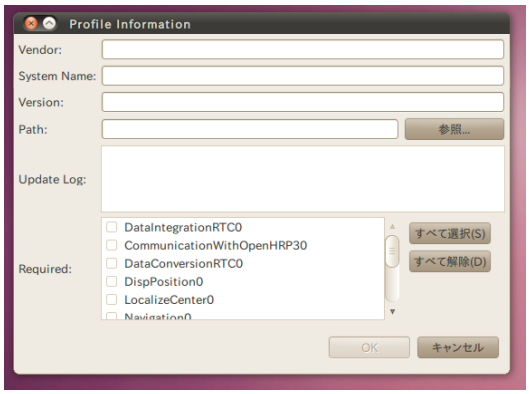


図 4.5.5　System Diagram の保存

Vendor には自分の名前を半角英字で，System Name には本システムに名付ける任意の名前を（ThreeWheeledRobotSystem など），Version には任意のバージョン（1.0.0 など）を，Path には任意の場所において xml の拡張子も含めた任意の名称（/home/(アカウント名)/workspace/ThreeWheeledRobotSystem.xml など）を入力し，Update Log は空欄で，Required では「すべて選択」を押下し，下の「OK」を押下します．すると，Required で選択した RTC を，all.sh で RTC を実行した後 System Diagram 上の任意の位置で右クリックして「Open」を選択することで，図 4.5.3 の状態にすることができます．Name Service

View にある RTC を System Diagram 上へ移動する手間を省略できます.

なお Path を決定するとき,「参照」を押下しても Path を決定することはできません. 前述した通り, Path の空欄に直接任意の場所と xml の拡張子込みの名称を入力し指定して下さい.

「Open and Restore」や「Open and Quick Restore」でも開くことはできますが, コンフィグレーションのエラーが発生する場合があります.

（ システムエディタ（セーブ編） | OpenRTM-aist：
http://openrtm.org/openrtm/ja/content/%E3%82%B7%E3%82%B9%E3%83%86%E3%83%A0%E3%82%A8%E3%83%87%E3%82%A3%E3%82%BF%EF%BC%88%E3%82%BB%E3%83%BC%E3%83%96%E7%B7%A8%EF%BC%89 ）

## ４．６　コンフィグレーションの変更

3.10.3 章の方法を用いて, RTC のコンフィグレーションの変更を行います. 3.10.3 章の方法とは,

(1) System Diagram 上でコンフィグレーションを変更したい RTC を選択
(2) RTC が黒枠で囲われ, Configuration View で「default」にチェックがついていることを確認
(3) 指定されたコンフィグレーションを選択し,「適用」を押下

という一連の手順のことを指します.

来訪者受付システムの RTC 群のコンフィグレーションについては, 3.5.1 章でダウンロードしたそれぞれのモジュール機能仕様書に詳細が記載されております. 適宜参考にして下さい.

本学習環境では, まず, コンフィグレーションのラジオボタンは以下の通りとします.

「default」: CommunicationWithOpenHRP3, DispPosition, SwitchInputRTC, PathFollower, SerialPortRTC, DataConversionRTC
「ThreeWheeledRobot」: PathPlanning, Navigation
「Simu_ThreeWheeledRobot」: Odometry
目標地点の座標の NodeID : PositionInput
出発地点の座標の NodeID : LocalizeCenter

つまり, PathPlanning, Navigation, Odometry, PositionInput, LocalizeCenter のみコンフィグレーションの設定を行って下さい. それ以外は最初から「default」が選択されており, 正しい設定となっています.

次に, コンフィグレーションの数値の変更を行います. 表 4.6.1 に従い PathFollower のコンフィグレーションの修正を行って下さい.

表 4.6.1　PathFollower のコンフィグレーション値の修正

| 名称 | 型 | デフォルト値 | 変更後の値 |
|---|---|---|---|
| control_cycle | double | 0.02 | 0.001 |
| max_acc_w | double | 0.1 | 0.3 |
| max_v | double | 1.0 | 0.4 |
| max_w | double | 0.3 | 1.0 |

これで，RTC のコンフィグレーションの変更は終了です.

## ４．７　ロボット演習

ロボットの電源が ON であることを確認し，図 4.7.1 のように All Activate のボタンをクリックすると，図 4.7.2 のように新しいウインドウが 2 つ現れます.

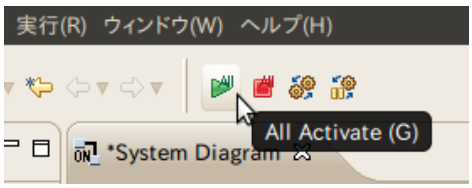


図 4.7.1　All Activate

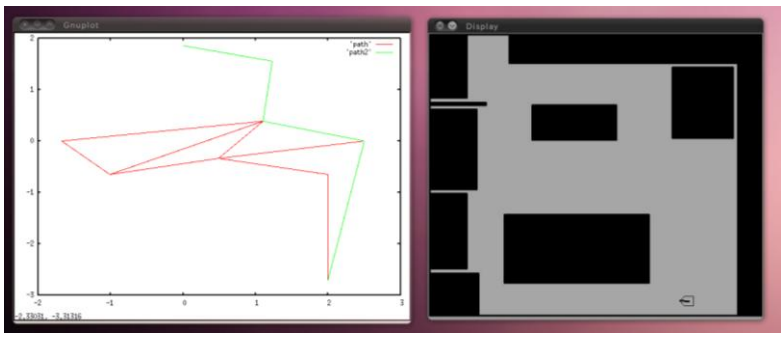

図 4.7.2　Activate 後に出現するウインドウ群の例

これで現在，DataConversionRTC と CommunicationWithOpenHRP3 以外の RTC はすべて Activate（緑色）となっています.

次に，先ほど GrxUI で開いたプロジェクト「ThreeWheeledRobot.xml」に戻ります．そして，図 4.7.3 のように Start Simulation をクリックします.

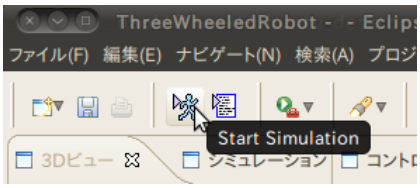


図 4.7.3　Start Simulation

すると，図 4.7.4 のようにログのクリアを確認されます．「OK」を押下して下さい．

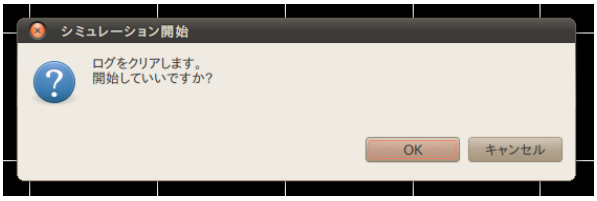


図 4.7.4　ログのクリアの確認

また，シミュレーションスタートが 2 度目以降の場合，図 4.7.5 のようにコントローラの再スタートも確認される場合があります．これは「はい」を押下して下さい．

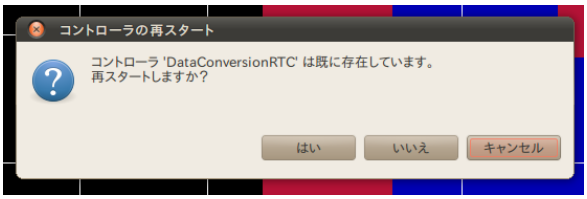


図 4.7.5　コントローラの再スタートの確認

これでシミュレーションが始まります．シミュレーション実行中においては図 4.7.6 のように，OpenHRP3 のシミュレーション環境と DispPosition によるロボットの移動した軌跡と現在位置が変化します．

実環境における 3 輪移動ロボットも，シミュレーションと同期し駆動します．なお，3 輪移動ロボットと PC を接続するケーブルが地面と接触している場合，ケーブルと床との摩擦が原因で 3 輪移動ロボットが想定通りの動作をしない場合があります．よって 3 輪移動ロボットを駆動させる際には，ケーブルを持ち上げ 3 輪移動ロボットの動作に影響がないようにして下さい．

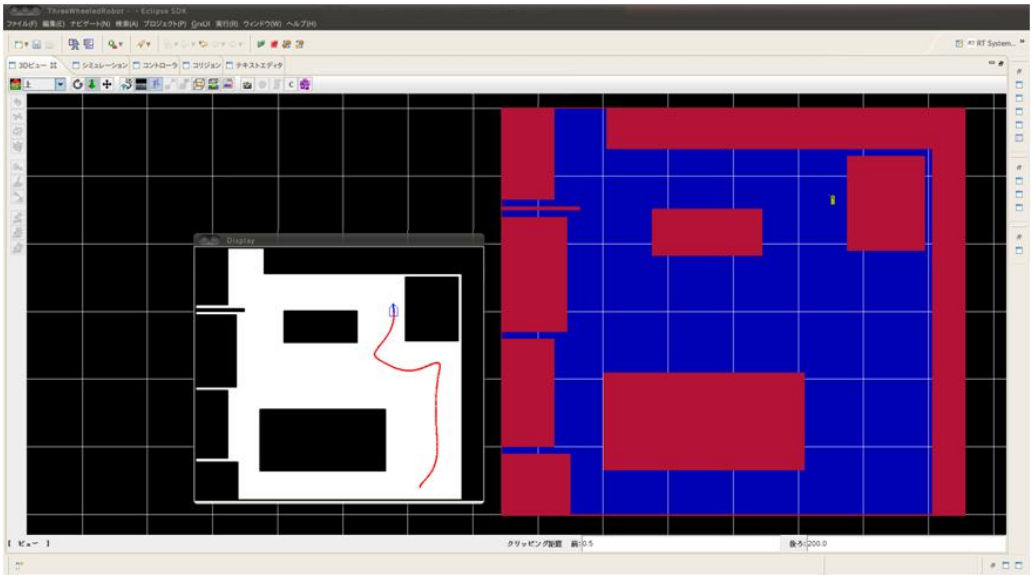

図 4.7.6　シミュレーション実行中の例

また，シミュレーションをスタートすることで，RT System Editor では，DataConversionRTC と CommunicationWithOpenHRP3 と DataIntegrationRTC が Acticate（緑色）になります．

図 4.7.7 のように目標地点に到着し，ロボットモデルと 3 輪移動ロボットが動作停止し次第終了です．また，その他の状況においてロボットモデルや 3 輪移動ロボットが停止しても，終了して下さい．

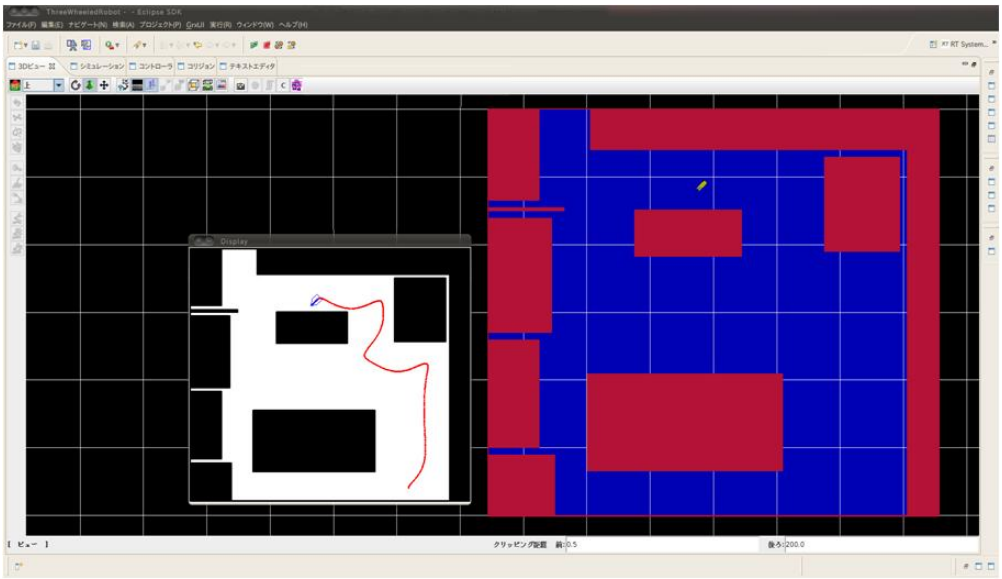

図 4.7.7　シミュレーション終了時の例

このとき，図 4.7.8 のようにシミュレーション終了をクリックすることで，図 4.7.9 のようにシミュレーション時間が表示され終了します．

実機とシミュレーションがうまく同期していない場合，図 4.7.9 のように「(B)/(A)」が大きすぎる（目安：1.4 以上）可能性があります．その場合可能ならば，DataConversionRTC のコンフィグレーションの中の，PGain と DGain を 0.1 刻み程度で変化させ，同期するために適当な数値として下さい．数値を大きくするとシミュレーションのロボットモデルの速度が上がります．

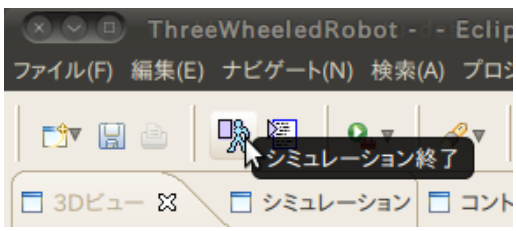


図 4.7.8　シミュレーション終了のボタン

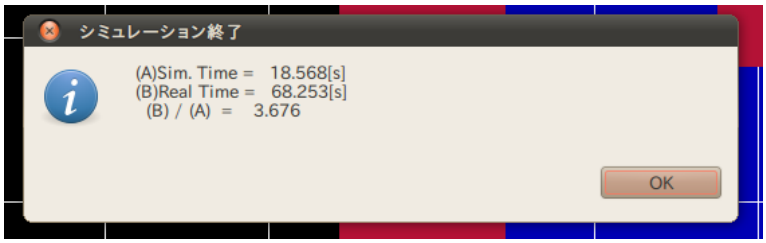


図 4.7.9　シミュレーション終了時の時間表示の例

RT System Editor に戻ると，DataConversionRTC，CommunicationWithOpenHRP3，DataIntegrationRTC が Deactivate になっています．他の RTC も Deactivate にするために，図 4.7.10 のように All Deactivate のボタンをクリックします．

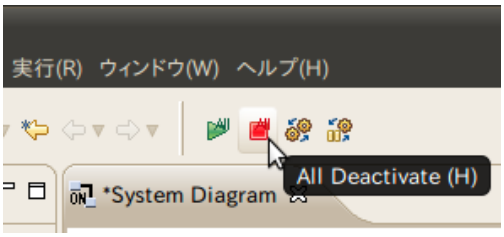


図 4.7.10　All Deactivate

これで，すべて Deactivate になり，ロボット演習は終了です．

もう一度シミュレーションを実行する場合は，目標地点などのコンフィグレーションを適切に設定した後，4.7 章ロボット演習を再度始めて下さい．

### ４．８　ログの確認

プログラム内にて，「RTC_DEBUG((“何かしらの情報”))」や「RTC_INFO((“何かしらの情報”))」のような書式で記されているものについては，生成されたログ内で確認することができます．また，それ以外の情報もログ内には記されているため，エラーが発生したときなどの情報の収集に役立ちます．

来訪者受付システムに関しては，/opt/RH/(それぞれのディレクトリ)/log にて RTC ごとのログが保存され確認できます．

SerialPortRTC，DataConversionRTC，DataIntegrationRTC に関しては，それぞれのディレクトリ内にある「rtc***.log」にてログが確認できます．

また，ログレベルやログの命名方法など，ログに関する規則は変更することができます．以下の URL を参考にログに関する規則を変更して下さい．

（　設定ファイル（基礎編）| OpenRTM-aist：
http://openrtm.org/openrtm/ja/content/%E8%A8%AD%E5%AE%9A%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB-%E5%9F%BA%E7%A4%8E%E7%B7%A8　）

# ５　おわりに

内容に問題点や疑問点があった場合，以下の連絡先にてお知らせ下さい.

E-Mail:openrtm@design.mech.saitama-u.ac.jp

または,

埼玉大学　設計工学研究室
〒338-8570
埼玉県さいたま市桜区下大久保 255
設計工学研究室
URL: http://design.mech.saitama-u.ac.jp/

# ６　参考 URL

【RTC ミドルウェア学習環境公開関連ページ】

・埼玉大学　設計工学研究室　「導入教育を目的とした安価で入手容易な RT ミドルウェア学習環境」公開 Web ページ.....4，10，34

http://design.mech.saitama-u.ac.jp/OpenRTM/

・OpenRTM-aist 公式 Web サイト「導入教育を目的とした安価で入手容易な RT ミドルウェア学習環境」公開 Web ページ.....4，10

http://www.openrtm.org/openrtm/ja/project/rtm_learning

・埼玉大学　設計工学研究室.....108

http://design.mech.saitama-u.ac.jp/

【OpenRTM-aist 関連ページ】

・OpenRTM-aist 公式 Web サイト.....6

http://www.openrtm.org/openrtm/ja/

・システムエディタ（セーブ編）｜OpenRTM-aist.....103

http://openrtm.org/openrtm/ja/content/%E3%82%B7%E3%82%B9%E3%83%86%E3%83%A0%E3%82%A8%E3%83%87%E3%82%A3%E3%82%BF%EF%BC%88%E3%82%BB%E3%83%BC%E3%83%96%E7%B7%A8%EF%BC%89

・設定ファイル（基礎編）| OpenRTM-aist.....107

http://openrtm.org/openrtm/ja/content/%E8%A8%AD%E5%AE%9A%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB-%E5%9F%BA%E7%A4%8E%E7%B7%A8

【OpenHRP3 関連ページ】

・OpenHRP3 の公式 Web サイト.....7，26

http://www.openrtp.jp/openhrp3/jp/index.html

・OpenHRP3 のインストールに関するページ.....7

http://www.openrtp.jp/openhrp3/jp/install.html

・OpenRTM-aist 公式 Web サイト内の OpenHRP3 公開 Web ページ.....7

http://openrtm.org/openrtm/ja/project/openhrp3

・OpenHRP3 >> コントローラブリッジ使用マニュアル.....18

http://www.openrtp.jp/openhrp3/jp/controller_bridge.html

・OpenHRP3 質問用掲示板の案内.....25

http://www.openrtp.jp/openhrp3/jp/forum.html

・OpenHRP3 Ver.3.1.1 ドキュメンテーショントップ.....26

http://www.openrtp.jp/openhrp3/3.1.1_3.0.8/jp/index.html