

### 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、本章をお読みください。

#### 3.1 OS の確認

本学習環境は、Ubuntu Linux 10.04 LTS でのみ動作確認を行なっています。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 の導入を行なってください。

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なってください。

#### 3.2 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/>

へ行き、左の「インストール (Ver.3.1.1)」を選択します。その後、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「ソースコードからインストール」か「バイナリパッケージからインストール」のうち好きな方を選択し、手順どおりインストールを行なってください。

途中で、「インストール手順」の下にある「GrxUI の起動・サンプルによる確認」を行うよう指示されると思います。この際、「GrxUI の起動・初期設定」の手順中でも指示されていますが、起動前に、

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

を忘れないようにしてください。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ください。

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ください。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-RELEASE」、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。

### 3.3 RTC の作成, ダウンロードなど

#### 3.3.1 Proxy RTC

ProxyRTC 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](http://ysuga.net/robot/rtm/practice/com_port)

を参考に作成します。

まず、3.2 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 (\*\*.sh) を実行し、Eclipse を起動します。すると workspace の選択画面となるため、問題がなければそのまま進みます。通常ならば、

**/home/(アカウント名)/workspace**

となるはずですが、この workspace の場所は覚えておいてください。

次に、図 7 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。

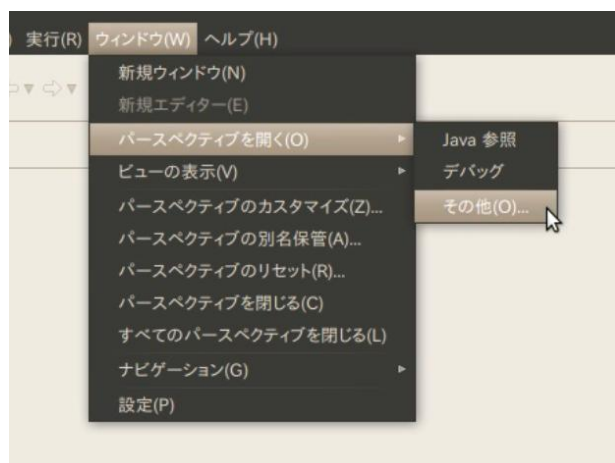


図 7 パースペクティブを開く

そして、その中から図 8 のように「RTC Builder」を選択します。



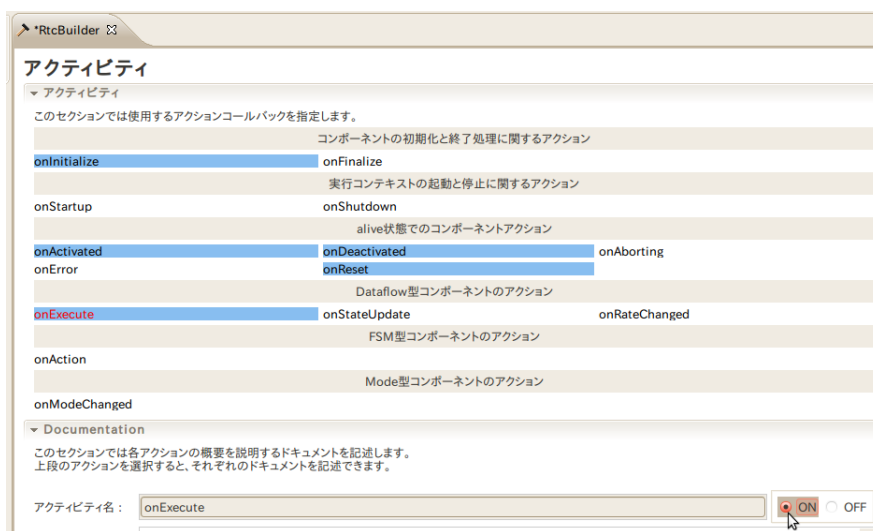


図 1 0 アクティビティ



図 1 1 データポート



図 1 2 コンフィギュレーション



図 1 3 言語・環境

項目を入力し終えたら図 1 4 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 1 5 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 1 6 の「**Generate success**」という生成終了のウインドウの表示を待ってください。生成し終わったら、Eclipse を終了して構いません。

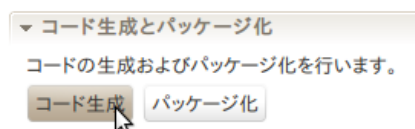


図 1 4 コード生成



図 1 5 プロジェクトの新規生成

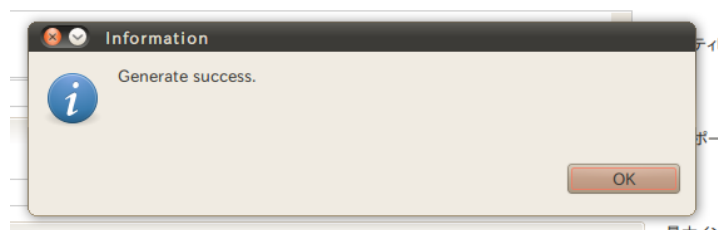


図 1 6 Generate success

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。（図 1 7）



図 1 7 端末の起動

次に、端末上で SerialPortRTC ディレクトリに移動します。

例えば、

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

のように、\$以降を入力することで移動することができます。これは Eclipse の初期起動時に自分で設定した workspace 内にある SerialPortRTC ディレクトリの場所に移動できるようにしてください。

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。編集する基本的な内容については「ysuga.net」の「5.1COM ポートにアクセスする RTC 作成」を参考にしてください。

ここからは、「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。ProxyRTC として実際に使えるように SerialPortRTC を編集します。

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（既に移動しているならば不要です）、

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

と入力して、ファイルの編集を行います。

### SerialPort.h の編集

13 行目を編集 (D)

```
#include <stdio.h>
```

この文章は、windows で ProxyRTC を使用する場合の include 文です。そのため、本学習環境の動作確認をした OS である Ubuntu Linux 10.04 LTS には使用されません。

### SerialPort.cpp の編集

18 行目以降を編集 (D)

```
#include <unistd.h>      //close,read,write
#include <termios.h>      //termios
#include <memory.h>       //memset
#include <fcntl.h>         //open,O_RDWR
#include <sys/select.h>    //fd_set,fds,FD_ZERO,dmy,FD_SET,FD_SETSIZE,select,FD_ISSET
#include <sys/ioctl.h>     //ioctl
#define _POSIX_SOURCE 1   //この行以降は変更なし
```

### SerialPortRTC.cpp の編集

136 行目に追加 (onExecute 内はじめ) (D)

```
    const unsigned int m_packet_size = 12;    //この行を追加
```

### 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。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 (既に移動しているならば不要です),

```
$sudo make -f Makefile.SerialPortRTC clean
```

```
$sudo make -f Makefile.SerialPortRTC
```

を入力し実行します。

これでエラーが発生しなければ、ProxyRTC の完成です。

### 3.3.2 Data Conversion RTC

DataConversionRTC はホームページの【RTC】よりダウンロードできます。

( 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/`

内にコピーしてください。

`$sudo cp -p -r (DataConversionRTC の場所) /usr/share/OpenHRP-3.1/sample/controller/`  
とすることでコピーできます。

### 3.3.3 Communication RTC (with OpenHRP3)

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。

### 3.3.4 Data Integration RTC

DataIntegrationRTC はホームページの【RTC】よりダウンロードできます。

ダウンロードした後, `workspace` 内にコピーしてください。

## 3.4 来訪者受付システムの導入

### 3.4.1 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://210.154.184.16/pukiwiki/?SYS\\_001\\_V100](http://210.154.184.16/pukiwiki/?SYS_001_V100)

にて, Download の【ソース】にある,

内容 : RH 関連, ファイル : RH\_10.zip

内容 : PA10 アーム, ファイル : PA10\_ARM.zip

をダウンロードしてください。

また, 【ドキュメント】にある,

内容 : RH 取扱説明書, ファイル : RH 取扱説明書 10\_05.pdf

をダウンロードしてください。

適宜, 以下の関連文書のダウンロードを行ってください。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

#### 【参考資料】

内容 : RH 関連(全部), ファイル : RH 関連参考資料.zip



### 3.4.2 来訪者受付システムの初期設定

3.4.1 章でダウンロードした RH 取扱説明書 10\_05.pdf を開きます。そして、5 ページの「3.3.3 OpenRTM-aist-1.0.0-RELEASE(C++)」の 2. から手順どおり行います。

#### 【注意点①】

OpenRTM-aist-1.0.0-RELEASE.tar.gz は、OpenRTM-aist 公式 Web サイト <http://www.openrtm.org/openrtm/ja/> の左にあるダウンロードから、OpenRTM-aist(C++版)の 1.0.0-RELEASE を選択し、ページ上部にある C++版ソースコードのから、「OpenRTM-aist-1.0.0-RELEASE.tar.gz」を選択しダウンロードしてください。

#### 【注意点②】

取扱説明書は OpenRTM-aist-1.0.0-RELEASE.tar.gz を「~/」にダウンロードしたときの手順なので、他の場所にダウンロードした場合は適宜修正してください。

#### 【注意点③】

手順 3. の「RH/patch」は存在しないため、3.4.1 章でダウンロードした「PA10\_ARM.zip」を展開した中にある「patch」ディレクトリ内のファイルで作業を行なってください。

#### 【注意点④】

全体的に改行がわかりにくく誤字もあるので、以下にわかりやすく書きます。  
手順 3.

```
cp ~/RH/patch/patch_RingBuffer.h.diff ~/OpenRTM-aist-1.0.0/src/lib/rtm
```

```
cp ~/RH/patch/Manager.cpp.diff ~/OpenRTM-aist-1.0.0/src/lib/rtm
```

手順 4.

```
cd OpenRTM-aist-1.0.0/src/lib/rtm
```

```
patch < RingBuffer.h.diff
```

```
patch < Manager.cpp.diff
```

手順 5.

```
/OpenRTM-aist-1.0.0$./configure --prefix=/usr
```

```
/OpenRTM-aist-1.0.0$make clean
```

```
/OpenRTM-aist-1.0.0$make
```

```
/OpenRTM-aist-1.0.0$sudo make install
```

#### 【注意点⑤】

手順 6. のコメントアウトは OpenHRP3 のインストール時にすでに行なっているはずなので、

不要です。

「3.3.3 OpenRTM-aist-1.0.0-RELEASE(C++)」が終了したら、次に、「3.3.7 OpenCV2.0」のインストールを行います。これも改行がわかりにくいので、以下にわかりやすく書きます。

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

「aptitude」の部分は、「apt-get」でもかまいません。

また、手順にはありませんが、gnuplot のインストールも行います

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

これで初期設定は終了です。次は RTC 群のダウンロード、修正、コンパイルです。

### 3.4.3 移動機能に関する RTC 群の修正

次は取扱説明書の 3.4 章ですが、多くの点で修正を行うため、以下の手順どおりに RTC 群の修正などを行ってください。

(1) zip ファイルの展開、コピー

3.4.1 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。

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

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) PCinRH ディレクトリの修正

PCinRH 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)

まず、以下のように移動してください。

```
$cd /opt/RH/PCinRH/src/comp
```

このディレクトリ内にある、mclean, mk, minst の修正を行います。

まず、バックアップを取ります。

```
$cp mclean mclean.org
```

```
$cp mk mk.org
```

```
$cp minst minst.org
```

拡張子の「org」は、オリジナルのファイルであることを示すためのものです。

そして、それぞれのファイルの編集を行います。

```
$gedit mclean mk minst
```

ここで 3 つのファイルとも、Urg\_to\_Obstacles の部分以外は削除してください。例として

mclean を載せます.

```
「mclean」
```

```
-----  
cd Urg_to_Obstacles  
make -f Makefile.Urg_to_Obstacles  
cd ..  
-----
```

3つのファイルで Urg\_to\_Obstacles の部分以外を削除したら, 修正は終了です.

## ②Urg\_to\_Obstacles ディレクトリ内の修正

Urg\_to\_Obstacles ディレクトリ内の修正を行うため, Urg\_to\_Obstacles 内に移動します.

```
$cd /opt/RH/PCinRH/src/comp/Urg_to_Obstacles
```

以下のように intellirobot.idl をバックアップし, コピーします.

```
$cp intellirobot.idl intellirobot.idl.org
```

```
$cp ../../../../idl/intellirobot.idl ./
```

次に, Urg\_to\_Obstacles.cpp の編集を行います.

```
$cp Urg_to_Obstacles.cpp Urg_to_Obstacles.cpp.org
```

```
$gedit Urg_to_Obstacles.cpp
```

80 行目, 81 行目を編集 (B)

```
double res = m_urg_res;  
double startAngle = floor(res * angleMin - 90.);
```

これで Urg\_to\_Obstacles ディレクトリ内の修正は終了です.

## ③Urg\_to\_Obstacles.conf の修正

conf ディレクトリに移動し, Urg\_to\_Obstacles.conf の修正を行います.

```
$cd /opt/RH/PCinRH/etc/conf
```

```
$gedit Urg_to_Obstacles.conf
```

6 行目を編集 (B)

```
logger.file_name:/opt/RH/PCinRH/log/Urg_to_Obstacles.log
```

これで Urg\_to\_Obstacles.conf の修正は終了です.

## (3) 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, RH2の部分のみ削除してください. 例として mclean を載せます.

```
「mclean」
```

```
-----  
cd CollisionDetection
```

```
make -f Makefile.CollisionDetection clean
```

```
cd ..
```

```
cd LocalizeCenter
```

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

```
cd ..
```

```
cd Navigation
```

```
make -f Makefile.Navigation clean
```

```
cd ..
```

```
cd ObstacleAvoidance
```

```
make -f Makefile.ObstacleAvoidance clean
```

```
cd ..
```

```
cd ObstacleMonitor
```

```
make -f Makefile.ObstacleMonitor 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 RH2 の部分を削除したら、修正は終了です.

②CollisionDetection ディレクトリ内の修正

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

nearest.h の修正を行います.

```
$cp nearest.h nearest.h.org
```

```
$gedit nearest.h
```

4行目を修正 (C)

```
#define ROBOT_RADIUS 0.2
```

③LocalizeCenter 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;
}

```

④ObstacleAvoidance ディレクトリ内の修正

\$cd /opt/RH/RHBase/src/comp/ObstacleAvoidance

nearest.h の修正を行います.

\$cp nearest.h nearest.h.org

```
$gedit nearest.h
```

4 行目を修正 (C)

```
#define ROBOT_RADIUS 0.2
```

⑤ObstacleMonitor ディレクトリ内の修正

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

nearest.h の修正を行います.

```
$cp nearest.h nearest.h.org
```

```
$gedit nearest.h
```

4 行目を修正 (C)

```
#define ROBOT_RADIUS 0.2
```

⑥Odometry 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
```

⑦SwitchInput 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;
}

```

#### ⑧\*\*.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, CollisionDetection.conf, LocalizeCenter.conf, Navigation.conf, NavigationParam.conf, ObstacleAvoidance.conf, ObstacleMonitor.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 3.5 章で別途解説します。

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

```
$gedit ~/.conf
```

まず, CollisionDetection.conf, LocalizeCenter.conf, Navigation.conf, ObstacleAvoidance.conf, ObstacleMonitor.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: 0.2
```

を付け加えてください. (C)

これは, 3 輪移動ロボットでの実機での計測値です.

また, NavigationParam.conf において, 最後に

```
## ThreeWheeledRobot ##
```

```
conf. ThreeWheeledRobot.judge_radius: 0.1
```

```
conf. ThreeWheeledRobot.max_vel: 0.3
```

を付け加えてください. (C)

#### (4) 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ください. 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。これは地図作成に関係しています。よって、3.5.5 章で別途解説を行います。

## ④ PathPlanning ディレクトリ内の修正

planner.cpp の修正を行います。これは地図作成に関係しています。よって、3.5.5 章で別途解説を行います。

## 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、DispPosition.conf, PathPlanning.conf, PathPlanningParam.conf, PositionInput.conf です。Position.conf も修正を行います。地図作成に関連しているため 3.5.5 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)

### 3.5 経路・経由点が記してある地図の作成

3.5 章では、地図画像などの作成を行います。自分で地図画像などを作成せず、著者の用意した環境を用いる場合は、ホームページ内の【シミュレーション環境】からダウンロードして展開し、それぞれ指示に従ってファイルの配置など行なってください。

#### 3.5.1 実環境の測定 (A)

現実に即したシミュレーション環境を作成するためには、現実の同様の環境の計測が不可欠です。従って、ロボットを走らせる部屋やフィールドなどの大きさ、障害物などの大きさをすべて計測してください。

#### 3.5.2 基本となる画像の作成 (A)

3.5.1 章で計測した実環境通りに、例（図 1 8）のように簡単な地図画像を作成してください。この地図画像は、Microsoft Office PowerPoint などでも長方形や円を組み合わせで簡単に作成することができます。

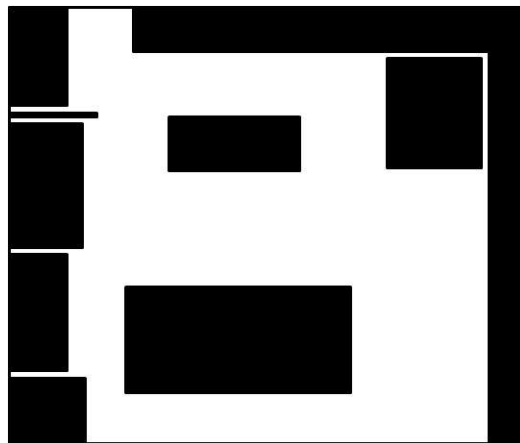


図 1 8 地図の基本となる画像例

この画像は、jpg 形式で `/opt/RH/RemotePC/bin/comp/` に、24 ビットの bmp 形式で `/opt/RH/RemotePC/src/tool/MapManager/src/` に適度な大きさ（目安としては縦横 600 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）で保存してください。

#### 3.5.3 MapEditor.java の修正

以下に移動します。

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

次に, MapEditor.java の修正を行います.

```
$gedit MapEditor.java
```

501 行目を編集 (C)

...“rtc-center.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)

mPerPix = 0.05; を, 自分の bmp の地図のピクセル当りの長さ[m]に変換します.

516 行目を編集 (C)

setOrigin(-262.0,-9.0); 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(0.0,0.0)で, 下方向が Y+, 右方向が X+, 座標の単位はピクセル数です.

修正は終了です. 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. (C)

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

これで MapManager を実行するための準備は完了です.

### 3.5.4 MapManager の実行

3.5.2 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
```

や, 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 自己位置姿勢推定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.

そして, 「FILE>SAVE」で route.dat などのファイル名で dat ファイルを適当な位置に保存します. 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.

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. これは, RTC「PathPlanning」の動作に必要な経路データです.

必須ではありませんが, 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「Print Screen」を押すなどして図のようにスクリーンショットを撮り, route\_map.png などのファイル名で /opt/RH/RemotePC/bin/comp/ に保存してください.

これで経路地図作成は終了です。MapManager を終了してください。

### 3.5.5 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, MapEditor.java で修正したように自分のファイルの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)

```
...+ 69;
```

```
...+ (333.0 - 63.0);
```

この最後の数字部分は、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。333.0 は y 方向の全ピクセル数であり、63.0 と 69.0 はそれぞれ x 方向, y 方向に平行移動した分のピクセル数です。

(まだ確定していない情報で申し訳ありませんが、座標軸はデフォルトでは画像右上隅が原点で、下方向が X+, 左方向が Y+です)

#### (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)

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ください。

#### (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]
```

```
...[-12:12]
```

これは実際に RT ミドルウェア学習環境を実行した際, 図 1 9 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, 初回時は調整せず, 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ください。

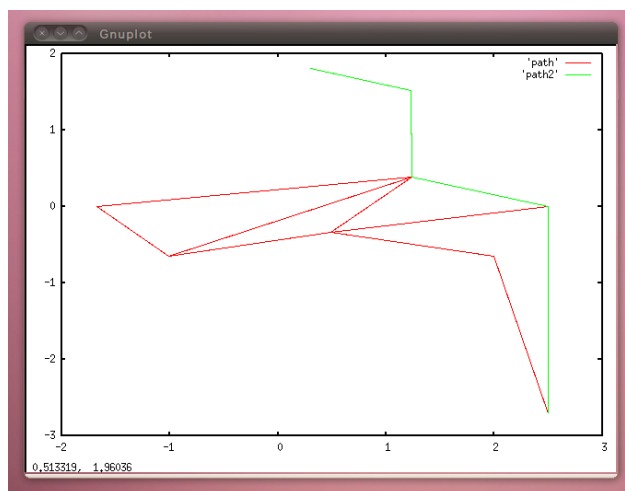


図 1 9 経路画像の例

#### (4) /opt/RH/RHBase/etc/conf/LocalPosition.conf

まず、MapManager 実行時に作成した route\_map と route\_map.png を開きます。

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

```
$eog route_map.png
```

```
$gedit route_map
```

次に、編集するファイルである LocalPosition.conf を開きます。RemotePC 内にも同盟のファイルが存在しますので注意してください。

```
$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ください。(C)

入力例

```
-----  
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
```

```
## start ##
```

```
conf.0start.startX: 2.50
conf.0start.startY: -2.70
conf.0start.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
```

-----

(5) /opt/RH/RemotePC/etc/conf/Position.conf

(4) と同じ手順で Position.conf も修正します。

まず, MapManager 実行時に作成した route\_map と route\_map.png を開きます。

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

```
$eog route_map.png
```

```
$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, LocalPosition.conf と同様の経路地図座標を入



力してください。 (C)

入力例

```
-----  
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
```

```
## start ##
```

```
conf.0start.01_x: 2.50  
conf.0start.02_y: -2.70  
conf.0start.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  
-----
```

### 3.6 シミュレーション環境の作成

3.6 章では、シミュレーション環境の作成を行います。自分で環境を作成せず、著者の用意した環境を用いる場合はホームページ内の【シミュレーション環境】からダウンロード

し、3.7 章に進んでください。

作成する方は、以下をお読みください。

### 3.6.1 シミュレーションモデルの作成 (A)

3.5.2 章で作成した地図画像と上から見た図が同様になるように、VRML ファイル (\*\*\*.wrl) を使用して表されるシミュレーションモデルを作成してください。

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](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>

にて開設されております。

また、3.6 章冒頭でホームページからダウンロードできる著者の VRML ファイルも参考にしてください。

作成したモデルは、/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保存すると実行時に手間が省けます。

### 3.6.2 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、3.2 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 (\*\*.sh) がある場所まで移動し、実行します。

(実行の例)

```
$ cd /home/(アカウント名)/eclipse
```

```
$sudo sh (作成したシェルスクリプト).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 2 0) で、GrxUI を選択し(図 2 1) 実行します。



図 2 0 パースペクティブを開く

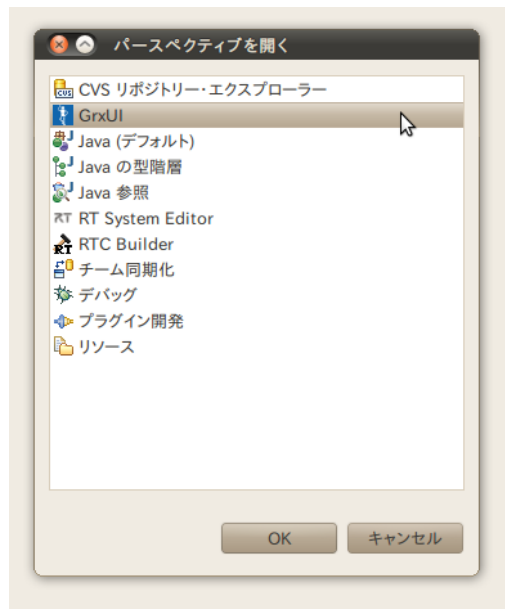


図 2 1 GrxUI の選択

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 2 2）。そこで、図 2 3 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 2 4）

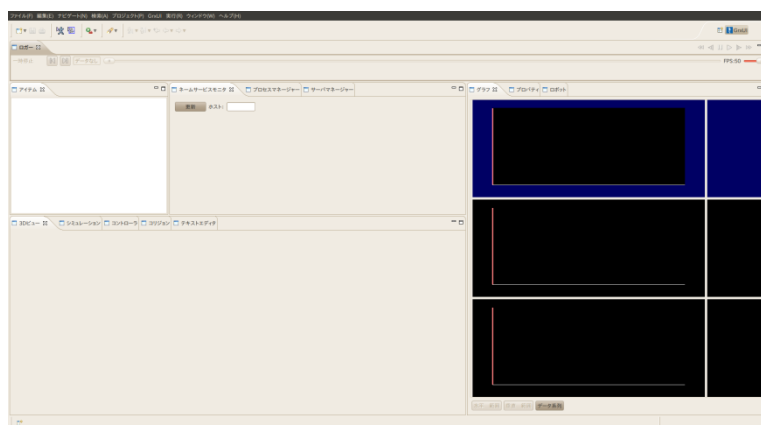


図 2 2 GrxUI でアイテムなどが表示されない場合



図 2 3 新規ウインドウを開く

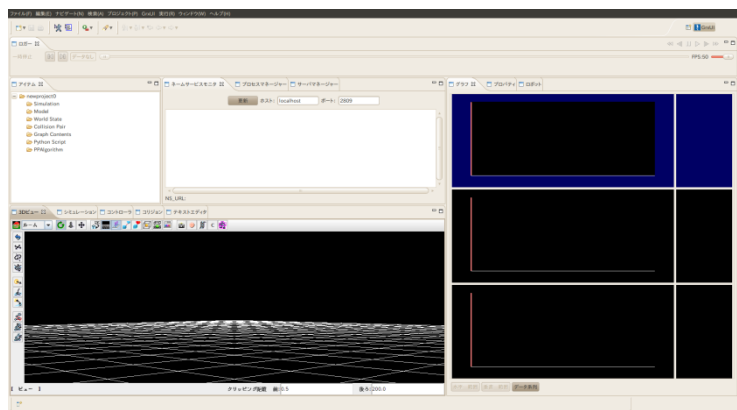


図 2 4 GrxUI でアイテムなどが表示されている場合

次に、プロジェクトの作成・保存を行います。

まず、図 2 5 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」を選択してください。すると、図 2 6 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 2 5 プロジェクトの作成

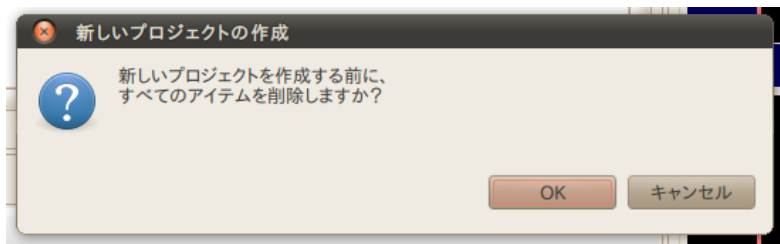


図 2 6 プロジェクトの新規作成に関する確認

次に、図 2 7 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」をしてください。これで、`newsimulation0` と `newworldstate0` が作成されます。



図 2 7 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、3.6.1 章で作成したモデルをすべて読み込んでください。すると、色や障害物の数、フィールドの広さなど多くの違いはありますが、図 2 8 のようなシミュレーション環境が構築されます。図 2 8 は上から見た図であり、3D ビューを最大化表示しているため表示方法に違いがあると思われます。

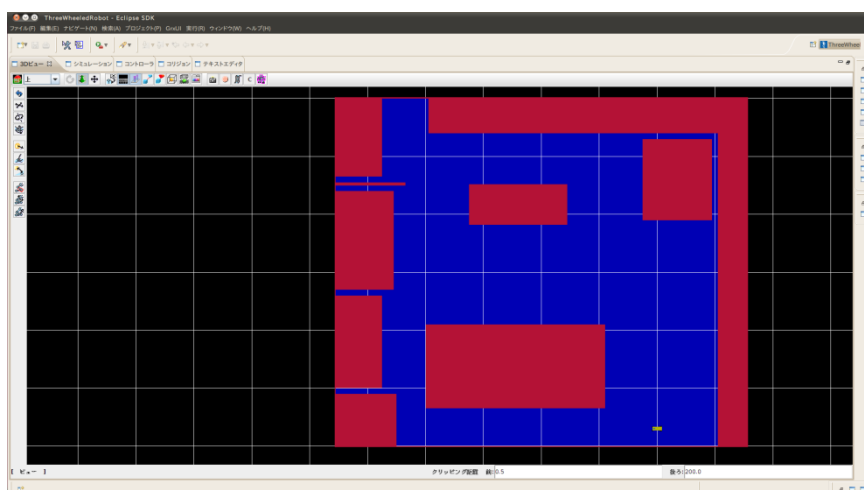


図 2 8 シミュレーション環境構築例

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」タブに移動してください。そこで、総時間は十分長く設定し（例えば 600）、積分時間は 0.002、ログ時間[秒]は 0.002、積分方法は RUNGE\_KUTTA、重力  $[m/s^2]$  は 9.8 とし、順動力学とビューシミュレーションにチェックをいれてください。

「コリジョン」タブに移動してください。「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。本学習環境において著者の環境を使用する場合は、表 1 2 のように設定してください。

表 1 2 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	0.8	0.8	0.001
MODEL_CAR	RIGHT_WHEEL	MODEL_FLOOR	floor	0.8	0.8	0.001
MODEL_CAR	LEFT_WHEEL	MODEL_FLOOR	floor	0.8	0.8	0.001
MODEL_CAR	CASTER_WHEEL	MODEL_FLOOR	floor	0	0	0.001
MODEL_CAR		OBSTACLE_BOX01		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX02		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX03		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX04		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX05		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX06		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX07		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX08		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX09		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_BOX10		0.5	0.5	0.01
MODEL_CAR		OBSTACLE_WALL		0.5	0.5	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 2 9 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「ThreeWheeledRobotProject.xml」など適切な名前で保存してください。

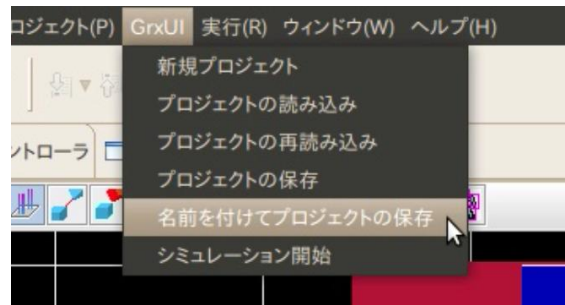


図 2 9 名前を付けてプロジェクトの保存

### 3.7 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。

#### 【PCinRH】

```
$cd /opt/RH/PCinRH/src/comp
```

```
$sh mclean
```

```
$sh mk
```

```
$sh minst
```

#### 【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。

### 3.8 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ください。

「start1.sh」

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

PCinRH_RTC_DIR=/opt/RH/PCinRH/bin/comp
PCinRH_CONF_DIR=/opt/RH/PCinRH/etc/conf
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=${PCinRH_RTC_DIR}    -t
Urg_to_Obstacles                -e                "/Urg_to_ObstaclesComp                -f
${PCinRH_CONF_DIR}/Urg_to_Obstacles.conf" ¥
    --tab-with-profile=Default    --working-directory=${RHBase_RTC_DIR}    -t
CollisionDetection              -e                "/CollisionDetectionComp              -f
${RHBase_CONF_DIR}/CollisionDetection.conf" ¥
    --tab-with-profile=Default    --working-directory=${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
ObstacleAvoidance              -e                "/ObstacleAvoidanceComp              -f
${RHBase_CONF_DIR}/ObstacleAvoidance.conf" ¥
    --tab-with-profile=Default    --working-directory=${RHBase_RTC_DIR}    -t
ObstacleMonitor                -e                "/ObstacleMonitorComp                -f
${RHBase_CONF_DIR}/ObstacleMonitor.conf" ¥
    --tab-with-profile=Default    --working-directory=${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」例

```

#!/bin/sh

cd /usr/share/OpenHRP-3.1/sample/controller/DataConversionRTC
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 3.5 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」例

```

#!/bin/sh

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

```

最後に、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」例

```
#!/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
```

#### 【注意点①】

\*\*sh の作成方法は、3.3.1 章 ProxyRTC 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。

### 3.9 3 輪移動ロボットの製作 (A)

3.9 章では、ロボットの製作を行います。シミュレーション環境のみを使用し、ロボットの製作を行わない場合は 4 章へ進んでください。

#### 3.9.1 部品表

部品表はホームページの【3 輪移動ロボット】よりダウンロードできます。

#### 3.9.2 電子回路図

電子回路図はホームページの【3 輪移動ロボット】よりダウンロードできます。

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/> )

#### 3.9.3 ガーバデータ

ガーバデータはホームページの【3 輪移動ロボット】よりダウンロードできます。

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。

#### 3.9.4 PIC プログラム

PIC プログラムはホームページの【3 輪移動ロボット】よりダウンロードできます。  
また、PIC の開発環境を表に示します。

表 1 3 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

#### 3.9.5 製作のための参考画像

3 輪移動ロボットの製作のために、参考画像を図 3 0～図 3 3 に示します。

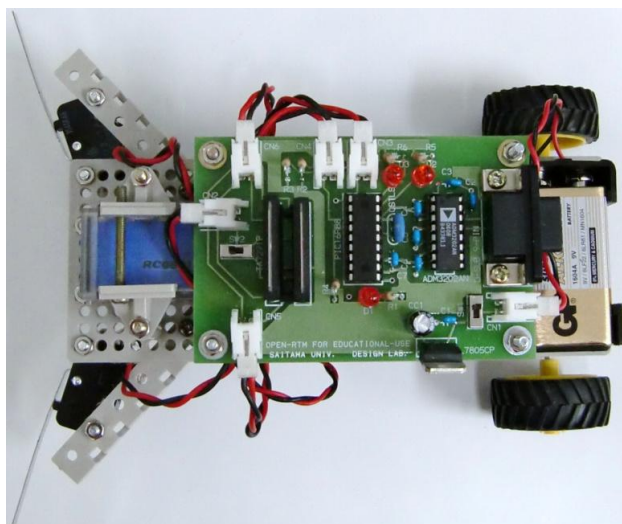


図 3 0 上方向からのロボット

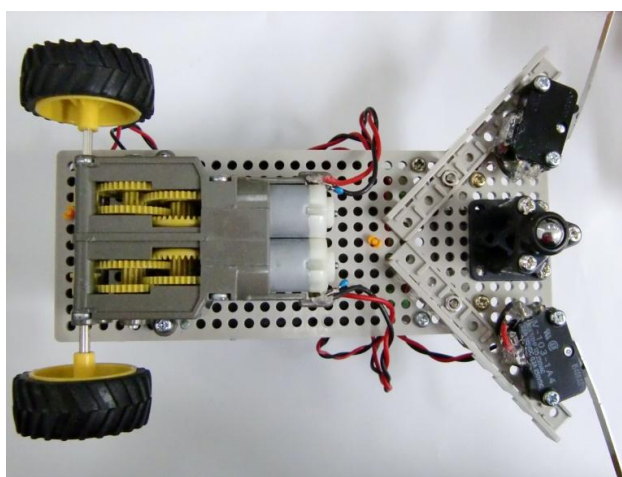


図 3 1 下方向からのロボット

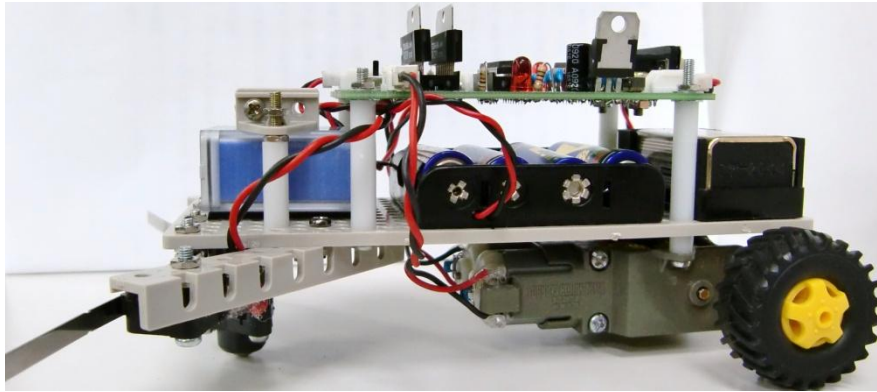


図 3 2 横方向からのロボット

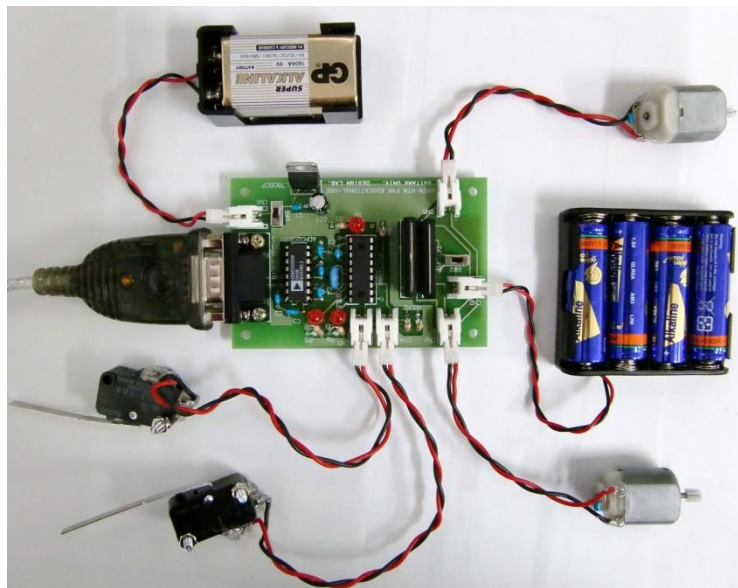


図 3 3 ロボットの制御回路

### 3.9.6 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，3.9 章でダウンロードできるデータは自由に変更していただいてもかまいません．