メカナムローバーVer2.1
ROS 制御モード

取扱説明書
（2020.08.04）

Vstone®
ヴイストン株式会社

このたびは、メカナムローバーVer2.1 をお買い求めいただき、誠にありがとうございます。本ドキュメントをよくお読みいただき、操作を行ってください。

目次

1　はじめに / 注意事項

本書は、メカナムローバーVer2.1 を ROS で制御する際の取扱説明書兼チュートリアルです。メカナムローバー本体についての説明書である「メカナムローバーVer2.1 取扱説明書」も併せてご確認いただき、注意事項を守り、安全に十分配慮してご使用ください。

本製品の使用にあたっては下記注意事項に従い、正しくご使用ください。

- ○　本製品を無許可で複製、再配布、再販することはできません。ただし、著作権法で認められた範囲における複製については許可されます。
- ○　本製品の対応環境は、ネイティブの Ubuntu 16.04 と ROS Kinetic または、Ubuntu 18.04 と ROS Melodic です(シミュレーション機能は ROS Melodic のみ対応)。それ以外の環境での使用についてはサポート対象外となる他、それによって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。
- ○　本製品の制御を仮想環境上の ROS から行われた場合、タイムラグ等により安全な制御が行えない可能性があります。そのため、仮想環境からの制御は非推奨とさせていただいており、その使用によって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。
- ○　本製品の使用にあたっては、本製品に含まれない公開ライブラリを多数使用する必要がございます。本製品に含まれないライブラリについてはサポート対象外となる他、その使用によって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。
- ○　本製品を使用中にハードウェアに強い振動や衝撃が加わると、暴走に繋がる可能性がございます。衝突などによる強い振動や衝撃を加えないでください。
- ○　本製品を使用する PC はお客様にてご準備ください。Ubuntu のデバイスドライバの対応状況等により、一部の機能が正常に動作しない可能性がございますが、デバイス固有の問題についてはサポート対象外となる他、それによって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。

2 ROS について

2.1 ROS の概要

ROS（Robot Operation System）は、OSRF（Open Source Robotics Foundation）によって開発・メンテナンスされているロボット用のミドルウェアです。分散処理が求められる複雑なロボットシステムを制御できる性能を備えており、世界中の研究者や開発者が作成した豊富なライブラリを使用することができます。ロボット制御システムの作成を効率化できることから、人型ロボットから車両型ロボット、水上・水中ロボットやドローンに至るまで、幅広い分野で活用されています。

ROS の特徴のひとつが、BSD ライセンスに基づくオープンソースとして公開されており、誰でも開発に参加し貢献できることです。ROS には強力な開発者コミュニティが存在し、誰でも使用可能な 5000 以上のライブラリのほとんどは、OSRF ではなくコミュニティによって開発・メンテナンスされています。

ROS 本体が BSD ライセンスによって提供されているため、ROS を用いて開発した成果物は、商用利用することが可能です。ROS を用いて動作する様々なロボットが発売されており、メカナムローバーもそのひとつです。ただし、ライブラリによっては BSD ではないライセンスによって提供されているものも存在するため、商用利用ではご注意ください。

2.2 ROS が提供する機能

ROS によって提供される主要な機能について、説明します。

- メッセージ通信

  ROS を用いて構成されるロボットシステムは分散処理が基本となっており、ユーザは「ノード」と呼ばれるプログラムを複数立ち上げることでシステムを作成します。例えば、ゲームパッドで操作できる台車ロボットであれば、「ゲームパッドの入力値を取得するノード」、「入力値に従って移動指令を出すノード」、「移動指令をもとにモータを回転させるノード」などが必要となります。当然、ノード間で情報を通信する仕組みが求められます。各ノード間で、センサ入力値の情報やカメラの映像、制御指令値といったデータをやり取りするために、ROS では Pub/Sub 方式によるメッセージ通信が提供されています。開発者はわずか数行のコードにより、任意の情報をパブリッシュ（配信）したり、サブスクライブ（購読）したりすることができます。メッセージには、構造体のような型が定められているため、ROS ノード同士であれば互換性を気にする必要はありません。また、型は自作することもできます。

- パッケージ管理

  ROS のライブラリやプログラムは、パッケージという単位で管理されています。パッケージの中にはノードやその設定ファイル、起動スクリプトなどが含まれており、ユーザは使用したい機能を持つパッケージをインターネット上からダウンロードし、ローカル環境に組み込むことができます。パッケージの追加や削除といった操作は非常に簡単に行う

事ができます。また、ユーザが独自の制御プログラムを開発する際には、まずパッケージを作成し、その中で開発を行う事になります。

- デバイスドライバ

  ROS では様々なデバイスのドライバがパッケージの形式で提供されています。対応しているデバイスであれば、パッケージを導入し、デバイスを接続するだけで使用することができます。

- ハードウェア抽象化

  ROS による制御システムは複数のノードによる分散処理によって動作します。これにより、ハードウェアが異なるロボットでも、上流の制御システムは同じものが使用できます。例えば未知環境の地図を作成する SLAM 機能を提供するパッケージ「gmapping」では、周囲の障害物の情報をレーザー光を用いた測距センサである LRF で取得することになっています。この LRF から情報を取得する部分は、LRF の種類に応じて異なるパッケージが提供されているため、ユーザは使用したい LRF を自由に選択することが可能です。そして開発者は、デバイス毎の違いを意識しなくても汎用的に使用可能な制御システムを作成することができます。

- ライブラリ

  ROS では、5000 を超える公開ライブラリを使用することができます。それらはデバイスドライバのようなものから、経路計画や動作生成といったものまで様々です。

- 視覚化ツール

  ROS には、いくつかの視覚化ツールが存在しており、ロボットの操縦 UI やデバッグ等のために活用されています。中でも「Rviz」は強力なツールです。3D 表示機能を持つこのツールは、実に多くのパッケージで情報を表示するために使われています。表示情報のカスタマイズが容易ですので、ユーザオリジナルの UI を作成することも容易です。

2.3 ROS に関する情報の集め方

ROS を用いた開発を行う際には、使用するパッケージの情報など、様々な情報が必要になります。ROS やその開発に関する情報は書籍から集めることもできますが、ここではインターネットから情報を集める際に参考になるサイトをいくつかご紹介します。

- ROS Wiki - http://wiki.ros.org/

  ROS の公式 Wiki です。ROS のインストール方法からチュートリアル、各公開パッケージの情報まで、様々な情報が公開されています。ただ、パッケージの情報などが更新されないまま古くなっていることもありますので、ご注意ください。

- ROS Wiki(ja) - http://wiki.ros.org/ja

  ROSWiki の日本語訳版です。現在も有志による精力的な翻訳作業が行われていますが、古い情報も多いので、英語版とあわせて確認した方がよいでしょう。

- ROS Answers - https://answers.ros.org/questions/

  ROS の Q&A フォーラムです。パッケージを使用した際のエラーの解消法など、様々な情報が蓄えられています。

3　メカナムローバーのセットアップ

メカナムローバー本体側のセットアップを行います。必ず「メカナムローバーVer2.1 取扱説明書」をご確認いただいたうえで作業を行ってください。シミュレーションを利用する場合はこの手順は必要ありません。次章の「ROS のインストール」にお進みください。

メカナムローバーを ROS で制御するためには、メカナムローバーの制御基板である VS-WRC051 を、USB 有線シリアルまたは、Wi-Fi を用いて PC 等の ROS が動作するデバイスと接続する必要があります。以下の手順に従って、メカナムローバー側のセットアップを行ってください。

① 「メカナムローバーVer2.1 取扱説明書」に従い、メカナムローバーの制御基板 VS-WRC051 を Arduino IDE で開発できるよう、環境をセットアップしてください。

② Arduino IDE メニューのファイル＞スケッチ例＞vs_wrc051_mecanumrover＞mecanumrover_common を開いてください。

③ スケッチ内の指示に従い、「WRC021_WIRELESS_MODE」,「WRC021_ROS_SERIAL_MODE」を適切に設定してください。また Wi-Fi を用いて接続する場合、メカナムローバーを接続するデバイスの IP アドレスおよび接続ポートを 69 行目以降の ROS 接続設定の項で設定してください。接続ポートは通常 11411 です。

④ setup()関数内のモータ制御パラメータ std_mortor_param[]について、必要に応じ各パラメータを適切に設定してください。

⑤ 更新したスケッチをメカナムローバーに書き込んでください。

スケッチ書き込み後、メカナムローバーが VS-C3 を使って動作することを確認してください。

※注意：Wi-Fi を用いて接続する場合、ROS との通信が確立するまで、VS-C3 を使った動作がスムーズに動作しません。ROS ライブラリの仕様です。

4 ROS のインストール

ROS Kinect または ROS Melodic を Ubuntu にインストールする方法を説明します。ここで述べる手順は、ROS Wiki で紹介されているインストール方法から一部を抜粋したものです。より詳しい情報が欲しい方は、下記ページを確認してください。

ROS Kinetic の Ubuntu へのインストール - http://wiki.ros.org/ja/kinetic/Installation/Ubuntu

ROS Melodic の Ubuntu へのインストール - http://wiki.ros.org/melodic/Installation/Ubuntu

※シミュレーションを利用する場合は、ROS Melodic をインストールしてください。

ROS のファイルはインターネットから取得するため、インターネット接続が必要です。PC をインターネットに接続してください。ネットワークの接続制限があったり、プロキシが設定されている環境では、ファイルのダウンロードに失敗することがあります。

① 端末（シェル）を立ち上げる

ランチャー上部の「コンピュータの検索」から“端末”を検索するか、ショートカットキー“Ctrl+Alt+T”を使用して、新しい端末（シェル）を起動します。

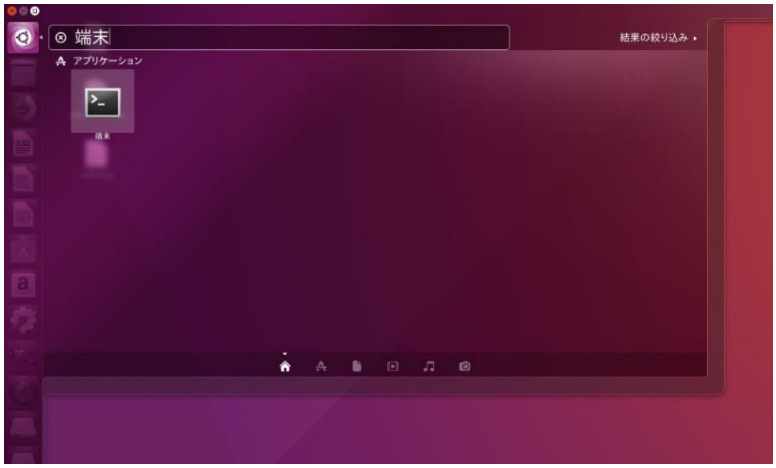

図 4-1 コンピュータの検索

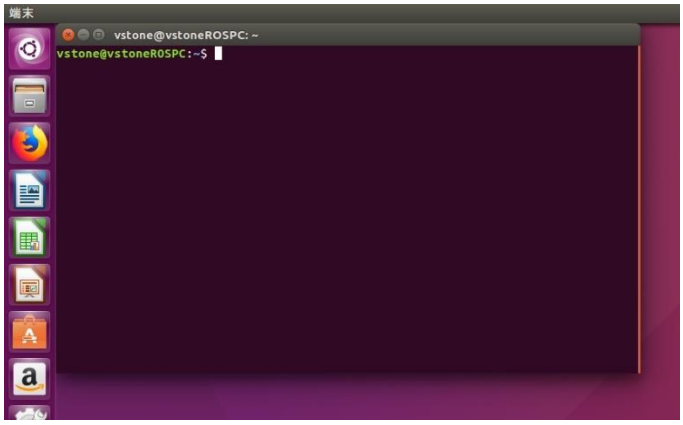

図 4-2 端末（シェル）

② sources.list を設定する

以下の青枠内のコマンドを端末にコピー&ペーストし、エンターキーを押して実行してください。コマンドラインへの貼り付けは右クリックまたは “Ctrl+Shift+V” で行えます。ひとつの青枠内にひとつのコマンドを書いていますので、複数行に分かれていてもまとめてコピーしてください。コピー&ペーストできない場合は手で入力してください。

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

③ 鍵を設定する

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

④ インストール

先にパッケージインデックスをアップデートしておきます。

```
sudo apt update
```

ROS Kinect をインストールします。ネットワーク環境によってはかなり時間がかかります。

```
sudo apt install ros-kinetic-desktop-full
```

ROS Melodic をインストールする場合は、以下のコマンドを使用してください。

```
sudo apt install ros-melodic-desktop-full
```

ROS のライブラリをバイナリでインストールする場合、上記のように「ros-バージョン-パッケージ名」で指定します。以下、バージョンに相当する箇所は、kinetic または melodic をご利用の環境に合わせて指定してください。

⑤ 環境設定

パスの設定を行います。Kinetic の場合、以下の 2 つのコマンドを順に実行してください。

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

Melodic の場合、コマンドは以下のようになります。

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

⑥　パッケージ構築のための依存ツールのセットアップ

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

⑦　rosdep の初期化

下記のコマンドで rosdep をインストールしてください。

```
sudo apt install python-rosdep
```

以下の 2 つのコマンドを順に実行してください。

```
sudo rosdep init
```

```
rosdep update
```

以上で、ROS Kinect 本体のインストールは完了です。ROS の基幹プログラムを立ち上げるコマンド“roscore”を使って、起動することを確認しておきましょう。

```
roscore
```

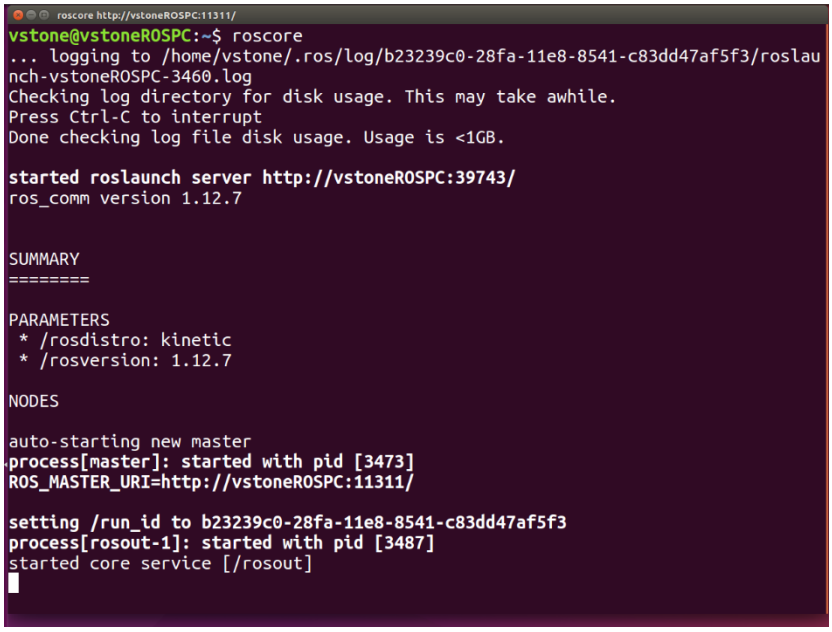

図 4-3 roscore 起動時のシェル

上のようなメッセージが表示されれば、ROS のインストールは正常に行われています。

roscore は、ROS の各サービスを提供する基幹プログラムです。ROS を使用する際には、必ずこの roscore を起動しておく必要があります。

5　メカナムローバー用サンプルパッケージのセットアップ

メカナムローバーを ROS で制御するサンプルパッケージのセットアップを行います。下記の手順に従って作業を行ってください。

5.1　catkin ワークスペースの作成

ROS のビルドシステムである catkin のためのワークスペースを作成します。この作業は、ROS Wiki に掲載されているチュートリアル「ROS 環境のインストールとセットアップ」（http://wiki.ros.org/ja/ROS/Tutorials/InstallingandConfiguringROSEnvironment）にも記載されています。

端末を起動して、以下のコマンドを順番に実行してください。

・フォルダの作成

```
mkdir -p ~/catkin_ws/src
```

・作成した src フォルダに移動

```
cd ~/catkin_ws/src
```

・ワークスペース作成

```
catkin_init_workspace
```

これでワークスペース「catkin_ws」ができあがりました。ワークスペースの src フォルダ内にパッケージフォルダを配置していくことができます。作成したばかりの src フォルダは空ですが、~/catkin_ws で以下のコマンドを実行することで、ワークスペースをビルドすることができます。

・(src フォルダに居る場合) ~/catkin_ws に移動

```
cd ~/catkin_ws
```

・ワークスペースをビルド

```
catkin_make
```

このワークスペース内で作成したパッケージを動作させるためには、ワークスペースをオーバーレイする必要があります。オーバーレイについては ROS Wiki にある catkin のチュートリアル「workspace_overlaying」(http://wiki.ros.org/catkin/Tutorials/workspace_overlaying)を確認してください。オーバーレイを行うためには、~/catkin_ws で次のコマンドを実行します。

```
source devel/setup.bash
```

ただしこの状態では、端末を新しく起動するたびにオーバーレイの作業を行う必要があります。端末起動時に自動的にオーバーレイが実行されるようにするためには、以下のコマンドを用いて設定を.bashrc に書き込みます。

```
echo "source /home/ユーザ名/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

以上で、ワークスペースの作成は完了です。

5.2 サンプルパッケージのダウンロードとビルド

GitHub で公開しているサンプルパッケージをダウンロードしビルドします。以下のコマンドを順番に実行してください。

・~/catkin_ws/src フォルダに移動します

```
cd ~/catkin_ws/src
```

・GitHub よりサンプルパッケージのソースコードをクローンします

```
git clone https://github.com/vstoneofficial/mecanumrover_samples.git
```

・ビルドします、表示が[100%]になればビルド完了です。

```
cd ~/catkin_ws
```

```
catkin_make
```

ROS の公開ライブラリの多くは GitHub でソースコードを公開しています。GitHub で公開されているものについては、同じ手順でワークスペースに追加、ビルドして使用することができます。

5.3 サンプルパッケージの内容について

メカナムローバーの ROS サンプルパッケージ「mecanumrover_samples」の内容について簡単に説明します。

- ノード

  mecanumrover_samples には、実機/シミュレーション双方で利用する「joycon」と「pub_odom」という 2 つのノードが存在します。「joycon」は、ゲームパッドなどのジョイスティックを使ってメカナムローバーを走行させられるノードです。「pub_odom」は、メカナムローバーのホイールエンコーダ値を基にオドメトリを出力するノードです。

  「vmecanum_」がファイル名の先頭に付くソースはシミュレーションに関するもので、実機の制御基板 VS-WRC051 上で実装された ROS ノードを再現しています。

  それぞれのソースコードは mecanumrover_samples/src に存在します。ソースコードを書き換えてビルドすれば、機能を変更することができます。

- launch ファイル

  launch フォルダ内に存在する拡張子が「launch」のファイルは、ROS で使用できる非常に便利なファイルです。ROS で制御システムを構築するためには数多くのノードを起動する必要がありますが、それをひとつひとつ手動でやるのは大変です。launch ファイルを使えば複数のノードを同時に起動することができます。

  他にも、パラメータの設定や remap 機能など、便利な機能を使用することができますので、積極的に利用しましょう。

  mecanumrover_samples の launch フォルダ内には、サンプルプログラムを実行するための複数の launch ファイルが格納されています。各 launch ファイルについては、7 章以降で説明します。

- configuration ファイル

  configuration_files フォルダ内に存在する、拡張子が「yaml」や「lua」のファイルです。サンプルプログラム実行時に使用する様々なパラメータ等について記載された設定ファイルです。

- map ファイル

  navigation サンプルで使用する map ファイルを格納するフォルダです。初期状態では参考用の map ファイルが入っています。

- CMakeLists.txt と package.xml

  CMakeLists.txt と package.xml は ROS パッケージに必須な、catkin の設定ファイルです。詳しくは下記のサイトを参照してください。

CMakeLusts.txt: http://wiki.ros.org/catkin/CMakeLists.txt

Package.xml: http://wiki.ros.org/catkin/package.xml

6　メカナムローバーと ROS の接続

メカナムローバー実機と ROS を接続し、メッセージ通信が行えるようにします。手順に従い作業を行ってください。シミュレーションを利用する場合は手順が異なるため「6.4 シミュレーションの実行」をお読みください。

6.1　rosserial のインストール

メカナムローバーの制御基板と PC は USB-シリアル通信で接続します。rosserial は USB-シリアル通信で接続したデバイスと ROS との通信をサポートするライブラリです。rosserial を使用することで、メカナムローバーは ROS のメッセージ通信に対応することができます。

以下のコマンドを実行して rosserial をインストールしてください。ROS バージョンには、kinetic または melodic を入力してください。

```
sudo apt install ros-ROS バージョン-rosserial
```

もしくは、rosserial をソースからビルドしてインストールすることもできます。

```
cd ~/catkin_ws/src
```

```
git clone https://github.com/ros-drivers/rosserial.git
```

```
cd ~/catkin_ws
```

```
catkin_make
```

6.2　urg_node のインストール

北陽電機製レーザーレンジファインダ URG-04LX-UG01 を使用するためには、urg_node パッケージをインストールする必要があります。urg_node は、SCIP 規格で通信を行う LRF のデータを ROS で活用するためのデバイスドライバの役割を果たします。

以下のコマンドを実行して urg_node をインストールしてください。

```
sudo apt install ros-ROS バージョン-urg-node
```

または、6.1 項と同様の手順で、以下のコマンドによりソースをクローンすることで、ソースからビルドしてインストールすることもできます。

```
git clone https://github.com/ros-drivers/urg_node.git
```

6.3　メカナムローバーと ROS の接続

作業を行う前に、メカナムローバーを十分充電しておいてください。メカナムローバーと PC を USB ケーブルで接続した後に、rosserial を使って通信を行います。以下の手順に従って作業を行ってください。なおこの作業は、メカナムローバーと ROS を接続するたびに行う必要があります。

〔制御基板と ROS の接続〕

メカナムローバーの制御基板 VS-WRC051 と ROS を接続します。

① VS-WRC051 のデバイス名の確認(有線シリアル使用時)

有線シリアル使用時には VS-WRC051 のデバイス名を設定する必要があります。Wi-Fi で接続する場合は手順③に進んでください。

本体前部に取り付けられたメカナムローバーの制御基板 VS-WRC051 と PC を USB ケーブルで接続します。続けて、端末で以下のコマンドを実行し、VS-WRC051 のデバイス名を確認します。

```
dmesg
```

```
554] usb 1-1: new full-speed USB device number 5 using xhci_hcd
759] usb 1-1: New USB device found, idVendor=0403, idProduct=6015
762] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
764] usb 1-1: Product: FT231X USB UART
766] usb 1-1: Manufacturer: FTDI
767] usb 1-1: SerialNumber: DO00B2HY
502] usbcore: registered new interface driver usbserial
516] usbcore: registered new interface driver usbserial_generic
526] usbserial: USB Serial support registered for generic
418] usbcore: registered new interface driver ftdi_sio
605] usbserial: USB Serial support registered for FTDI USB Serial Devi

656] ftdi_sio 1-1:1.0: FTDI USB Serial Device converter detected
838] usb 1-1: Detected FT-X
655] usb 1-1: FTDI USB Serial Device converter now attached to ttyUSB0
oneROSPC:~$
```

図 6-1　制御基板のデバイス名確認

図 6-2 がコマンドの応答例です。4 行目の記述に「FT231X USB UART」とあります。これは VS-WRC051 上の USB シリアル変換チップが認識されたことを示しています。VS-WRC051 の名称は応答に現れませんのでご注意ください。14 行目の記述のうち「ttyUSB0」が VS-WRC051 のデバイス名となります。

デバイス名は接続する PC や、接続する順番などによって変化します。誤ったデバイス名で操作すると、誤動作の原因となりますのでご注意ください。

② デバイスの権限の設定（有線シリアル使用時）

LRF と制御基板を接続しただけでは、権限が不足しているため ROS からアクセスすることができません。次のコマンドでパーミッション設定を変更し、アクセスできるようにします。なお、Wi-Fi 接続で制御する場合は、この手順は不要です。手順③に進んでください。

```
sudo chmod 666 /dev/VS-WRC051 のデバイス名
```

本書の例では、VS-WRC051 のデバイス名が「ttyUSB0」でしたので、コマンドは次のようになります。

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

③ rosserial の実行

rosserial を実行し、メカナムローバーと通信を行います。roscore を起動してから、別の端末（シェル）で以下のコマンドを実行してください。

・有線シリアル接続の場合

デバイス名には、先ほど確認した VS-WRC051LV のデバイス名が入ります。

```
rosrun rosserial_python serial_node.py _port:=/dev/デバイス名
_baud:=115200
```

ここで“_baud:=115200”は、通信速度「ボーレート」を表しています。単位は[bit/sec]です。数字が大きいほど早い通信速度となります。ボーレートに設定できる値には 9600, 19200, 115200 など何種類かありますが、実際に使用可能かどうかは PC 環境によって異なるので調べてください。また、ボーレートを変更する際には、メカナムローバーのスケッチもそれに合わせて変更する必要があります。

・Wi-Fi 接続の場合

IP アドレスの設定等は不要です。メカナムローバーが起動した状態で、以下のコマンドを実行してください。

```
rosrun rosserial_python serial_node.py tcp
```

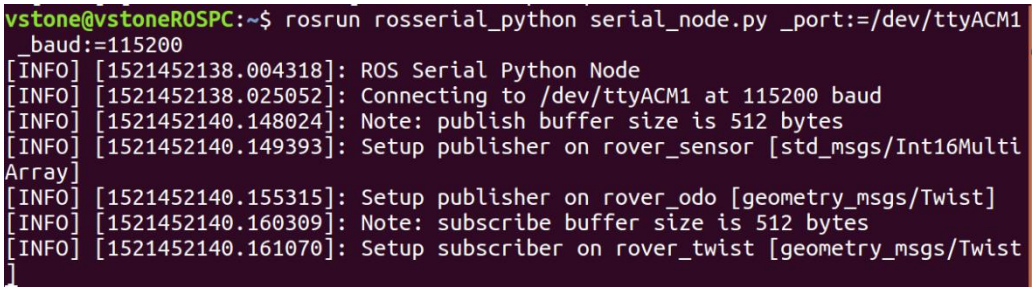

図 6-2　rosserial の応答（成功）

接続が成功した際の応答を図 6-3 に示します。これと同じ応答が得られれば、メカナムローバーと ROS の接続は成功です。

次に、エラーが発生したときの応答例を示します。

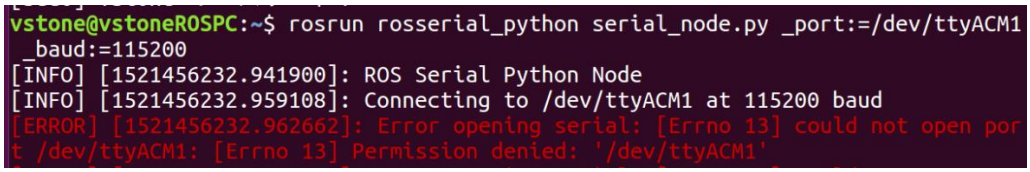

図 6-3　rosserial の応答（エラー1）

「Permission denied」は、パーミッションの設定が間違っていることを示しています。もう一度手順②でデバイス名を確認し、手順③を実施してください。

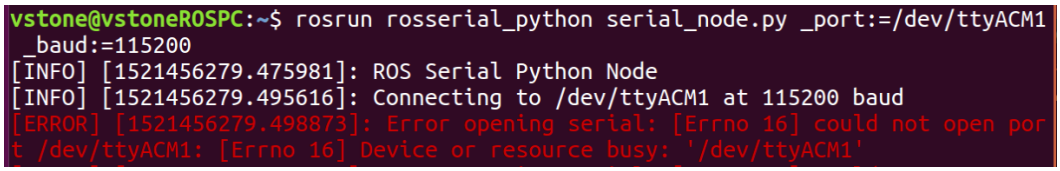

図 6-4　rosserial の応答（エラー2）

「Device or resource busy」は rosserial 以外のプロセスがデバイスにアクセスしている等により接続できないことを示しています。しばらくそのまま放置しておくことで、接続できることがあります。

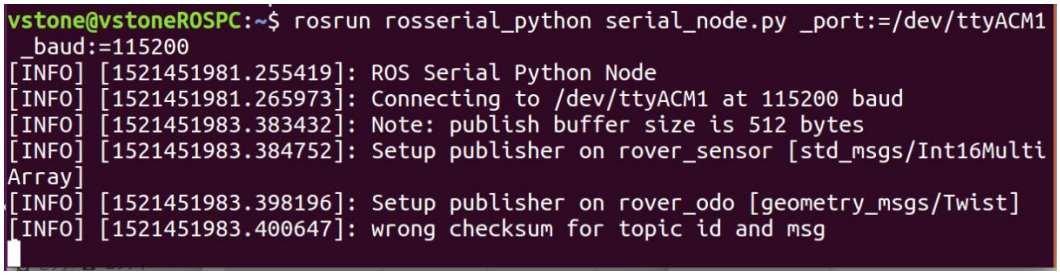

図 6-5　rosserial の応答（エラー3）

「wrong checksum for topic id and msg」は、メカナムローバーから受信したメッセージのチェックサムが一致しなかったときに生じるエラーです。**このエラーは発生したタイミングにより行うべき対処が異なります。**

- **通信開始時に発生したとき**

  メカナムローバーとの通信がうまく確立されていません。もう一度 rosserial を実行しなおしてください。ボーレートの値に問題がある可能性もあります。

- メカナム**ローバーを動作させている最中に発生したとき**

  通信中の一部のビットが欠落したことで発生したものと考えられます。メカナムローバーが正常に動作しているようであればそのまま操作を継続して問題ありません。ボーレートを変更することで改善する可能性があります。

```
[ERROR] [1553585292.859126]: Lost sync with device, restarting...
[INFO] [1553585292.861635]: Requesting topics...
[INFO] [1553585292.933221]: Setup publisher on rover_sensor [std_msgs/Int16Multi
Array]
[INFO] [1553585292.937173]: Setup publisher on rover_odo [geometry_msgs/Twist]
[ERROR] [1553585307.932183]: Lost sync with device, restarting...
[INFO] [1553585307.933795]: Requesting topics...
[INFO] [1553585308.005898]: Setup publisher on rover_sensor [std_msgs/Int16Multi
Array]
[INFO] [1553585308.009193]: Setup publisher on rover_odo [geometry_msgs/Twist]
```

図 6-6　rosserial の応答（エラー4）

「Lost sync with device, restarting…」は、メカナムローバーと ROS との通信がタイムアウトした際に発生します。タイムアウト時間は初期設定では 15 秒となっています。**このエラーはメッセージの内容により対処が異なります。**

- **Lost sync with device に続けて、Setup publisher が流れる場合**

  図 6-6 のように、Lost sync with device に続けて Setup publisher が表示される場合は、エラーではなく、メカナムローバーと ROS は正常に通信できている可能性が高いです。エラーを無視して作業を続けて頂いて構いません。

  rosserial のタイムアウトカウントは、メカナムローバー側で spinOnce()関数が呼び出されることでリセットされます。メカナムローバーでは仕様により、ROS からメッセージが配信されたときにのみ spinOnce()関数が呼び出されるため、メカナムローバーに速度指令値を送信しない状態が 15 秒以上続くと、タイムアウトが発生します。

- **Lost Sync with device のみが流れる場合**

  何らかの理由で接続に失敗しています。もう一度、接続手順を最初から実施指定ください。

〔LRF と ROS の接続〕

メカナムローバーに搭載されている LRF と ROS を接続します。LRF が搭載されていない場合、LRF を使用しないタスクの場合は、この手順は不要です。

メカナムローバーVer2.1 にオプションで搭載できる北陽電機製レーザーレンジファインダ URG-04LX-UG01 を例に、LRF の接続について説明します。

① LRF の接続とデバイス名の確認

本体前部に取り付けられた URG-04LX-UG01 と PC を、USB ケーブルで接続します。続けて端末で以下のコマンドを実行し、接続した LRF のデバイス名を確認します。

```
dmesg
```

```
2.620611] usb 1-2: new full-speed USB device number 7 using xhci_hcd
2.788631] usb 1-2: device descriptor read/64, error -71
3.079721] usb 1-2: New USB device found, idVendor=15d1, idProduct=0000
3.079735] usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
3.079744] usb 1-2: Product: URG-Series USB Driver
3.079752] usb 1-2: Manufacturer: Hokuyo Data Flex for USB
3.081432] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
e@vstoneROSPC:~$
```

図 6-7　LRF のデバイス名確認

図 6-1 がコマンドの応答例です。6 行目の記述に「Hokuyo Data Flex for USB」とありますので、URG-04LX-UG01 が認識されたことが分かります。7 行目の記述のうち「ttyACM0」が URG-04LX-UG01 のデバイス名となります。

デバイス名は接続する PC や、接続する順番などによって変化します。誤ったデバイス名で操作すると、誤動作の原因となりますのでご注意ください。

② デバイスの権限の設定

LRF と制御基板を接続しただけでは、権限が不足しているため ROS からアクセスすることができません。次のコマンドでパーミッション設定を変更し、アクセスできるようにします。

```
sudo chmod 666 /dev/URG-04LX-UG01 のデバイス名
```

本書の例では、URG-04LX-UG01 のデバイス名が「ttyACM0」でしたので、コマンドは次のようになります。

```
sudo chmod 666 /dev/ttyACM0
```

③　（任意）urg_node を用いた LRF の動作確認

接続した LRF の信号が正しく受信できているか確認を行います。なお、接続設定自体は手順②までで完了していますので、この手順は任意で実施してください。

urg_node は、SCIP 規格で通信を行う LRF のデータを ROS で活用するためのデバイスドライバの役割を担います。本製品に付属の LRF を使用したサンプルプログラムでは、launch ファイル呼び出し時に、urg_node を起動するよう記載しています。

roscore が起動した状態で、新たな端末で以下のコマンドを実行し urg_node を起動します。

```
rosrun urg_node urg_node _serial_port:="/dev/ URG-04LX-UG01 のデバイス名
```

本書の例では、URG-04LX-UG01 のデバイス名が「ttyACM0」でしたので、コマンドは次のようになります。

```
rosrun urg_node urg_node _serial_port:="/dev/ttyACM0"
```

続いて、Rviz を用いて LRF からの信号が正常に受信できているかを確認します。新しい端末で以下のコマンドを実行し Rviz を起動します。

```
rviz
```

Rviz が起動したら、下図の赤丸で示した「Fixed Frame」の設定値を「laser」としてください。設定値の部分をクリックすることで入力できるようになります。なお、利用状況等によっては Rviz の表示内容が異なることがあります。

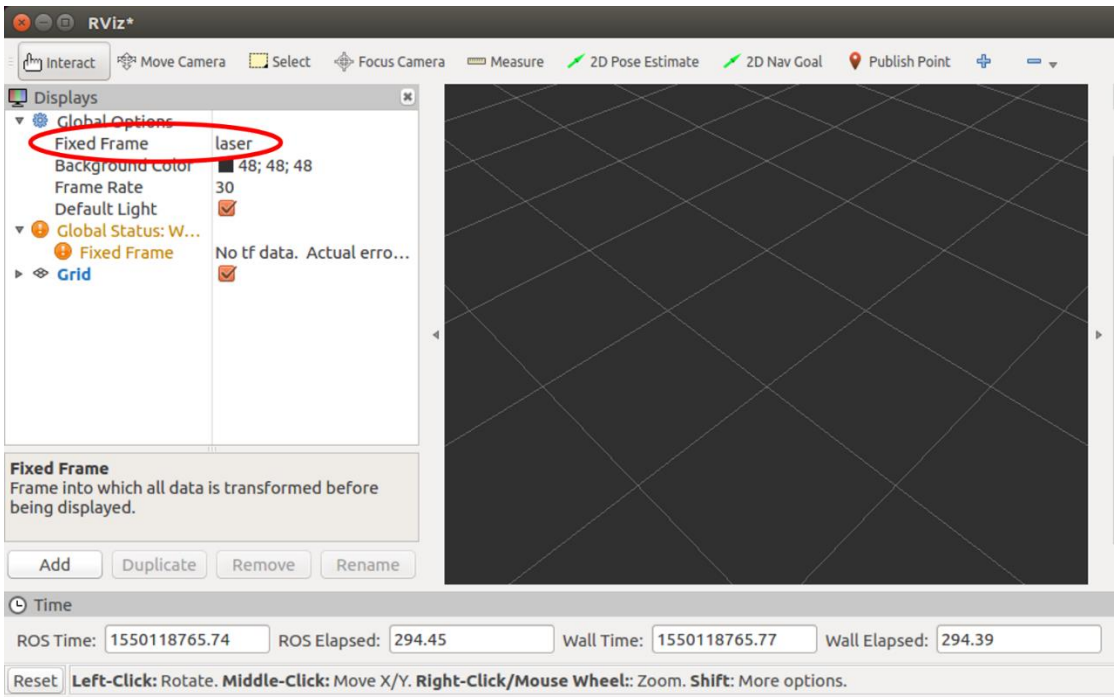

図 6-8　Rviz 起動画面

続いて左下の Add ボタンを押し、現れた選択画面から、「Laser Scan」を選択して OK ボタンを押して下さい。

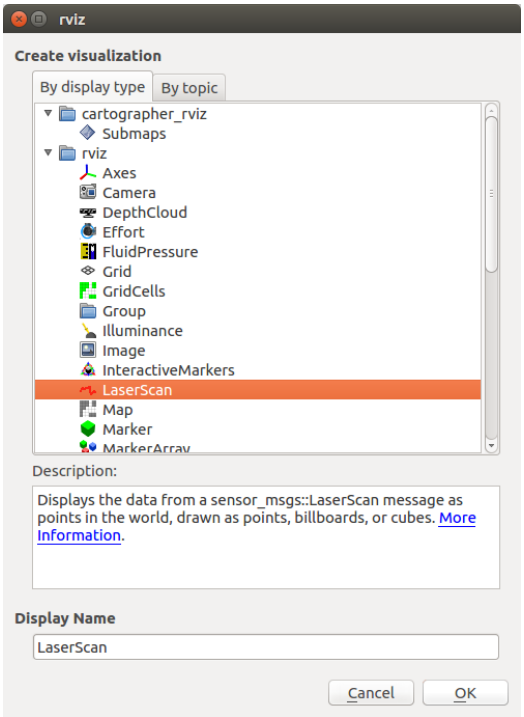

図 6-9　Create visualization 画面

Displays に Laser Scan が追加されたら、左端の三角印を押して詳細を開き、「Topic」の設定値を「/scan」に変更します。その後、下図のように LRF の捉えた障害物が点群として表示されることを確認してください。点群の見た目は異なる場合があります。

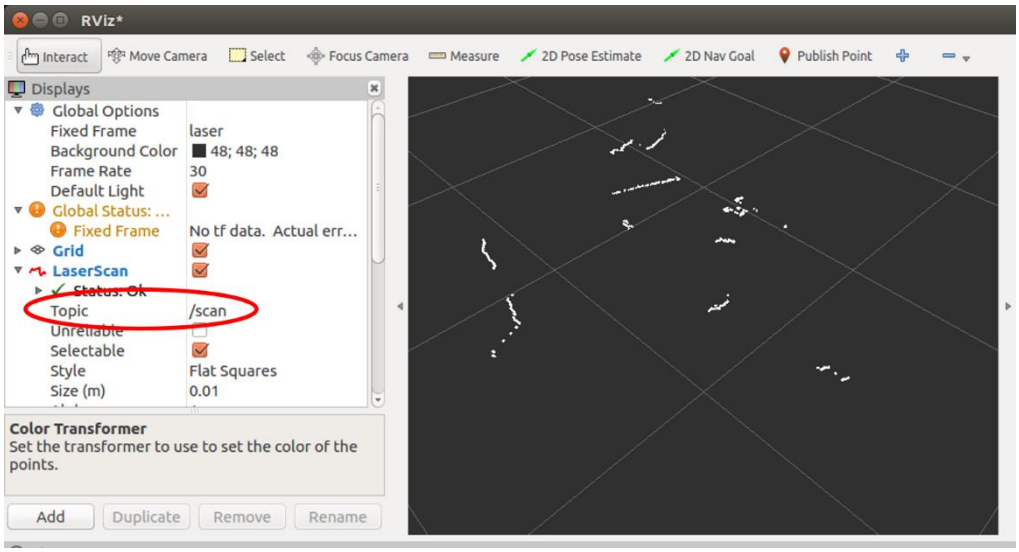

図 6-10　Rviz で LRF の測距情報を確認

6.4　シミュレーションの実行と概要

シミュレーションを利用する前に必要な jointcontroller をインストールします。下記のコマンドを実行してください。

```
sudo apt install ros-melodic-effort-controllers
```

シミュレーションを起動する場合は、専用の launch ファイルを実行します。下記のコマンドを実行してください。

```
roslaunch mecanumrover_samples vmecanumrover.launch
```

実行するとシミュレーション「gazebo」が立ち上がり、gazebo の画面内にシミュレーションのメカナムローバーが表示されます。

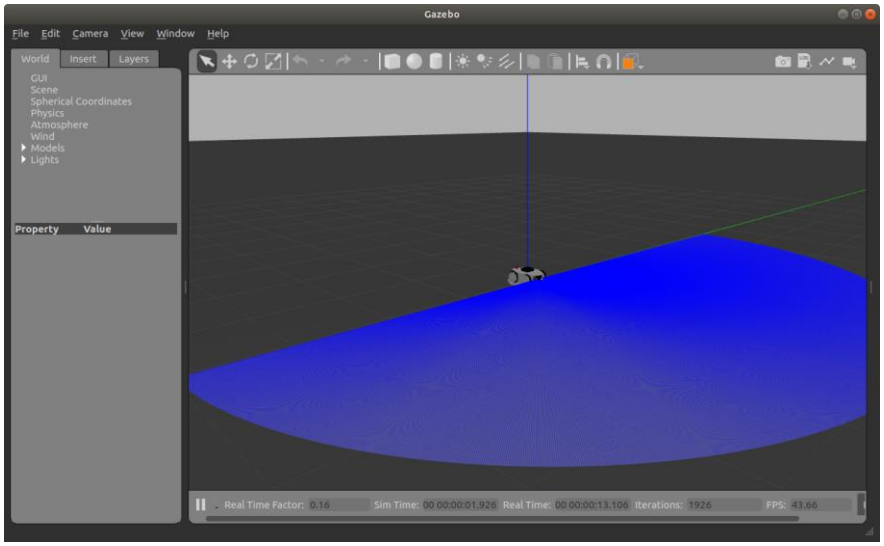

図 6-11　シミュレーションの起動画面例

① 画面操作

シミュレーションの画面の概要と操作について説明します。

中央の 3D 表示されたビューは、マウスでドラッグすることでカメラアングルを変更可能です。マウスの左・右・ホイールの各ボタンについて、アングルの視点の位置をクリックしてドッグすると、それぞれ視点の平行移動・拡大縮小・回転ができます。操作中はクリックした操作の起点位置に黄色のマーカーが表示されます。遠距離部分をクリックして操作すると意図せず急激に視点が変化するため注意して下さい。

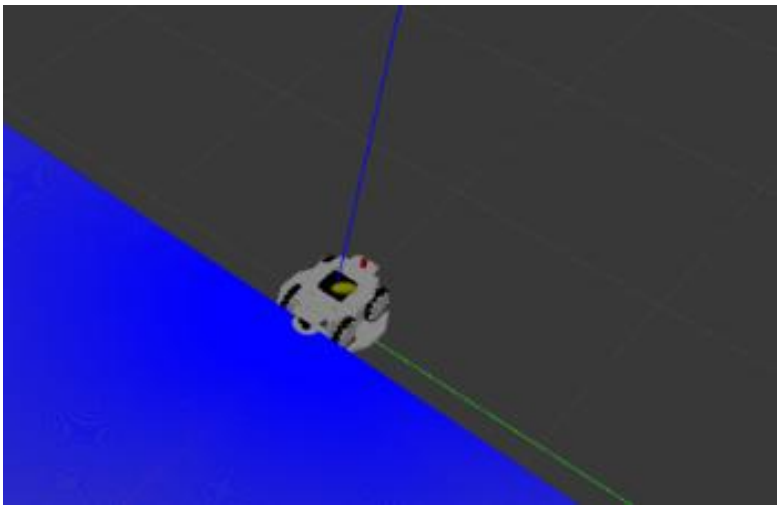

図 6-12　ビュー内をクリックして視点を操作(黄色がクリック位置)

画面下部には現在の実時間での経過時間(Real Time)・シミュレーション上の経過時間(Sim Time)・フレームレート(FPS)などが表示されます。後述の通り、起動後数秒間はサスペンションのリフトアップを行うため、Sim Time の経過時間を確認してください。
また、  をクリックするとシミュレーション内の時間経過を一時停止できます。もう一度クリックすると時間経過を再開します。

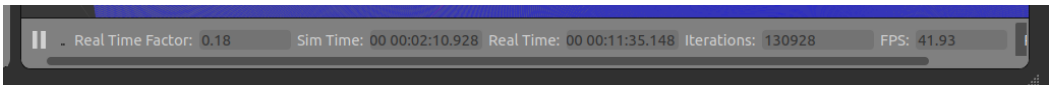

図 6-13　シミュレーションの経過時間や FPS の表示

ビューの上に配置されたツールバーからは、シミュレーション上の物体の位置・姿勢・サイズを変更したり、新たな物体を追加したりすることができます。

図 6-14　ツールバー

ツールバーの  の各ボタンは、物体の移動・回転・拡大縮小をそれぞれ行います。いずれかのボタンをクリックし、続いて操作する物体をビュー内でクリックすると、その物体に赤・青・緑の操作用のマーカー(インターフェース)が表示されるので、マウスでドラッグして変更してください。

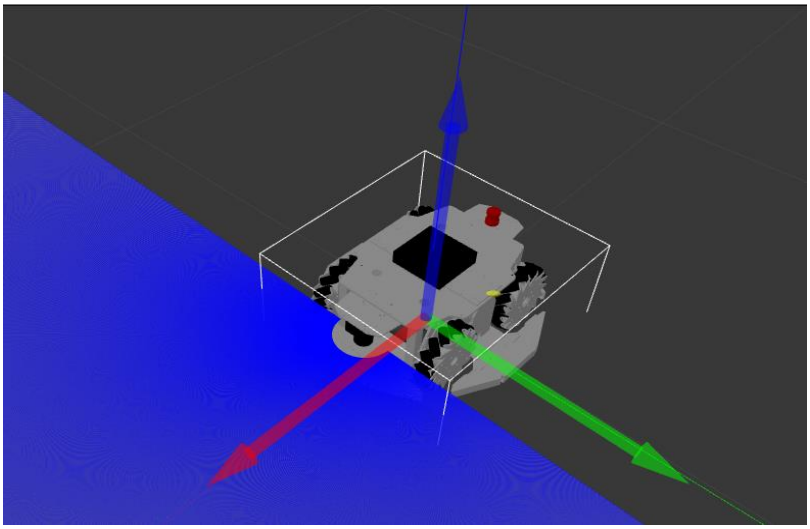

図 6-15　赤・青・緑のマーカーをドラッグして操作

ツールバーの  の各ボタンは、ビュー内に立方体・球・円柱を追加します。ボタンをクリックすると、ビュー内をマウスカーソルを追従するように追加する物体が表示されるので、追加する位置でクリックしてください。追加した物体は、前述の操作で移動・回転・拡大縮小が可能です。

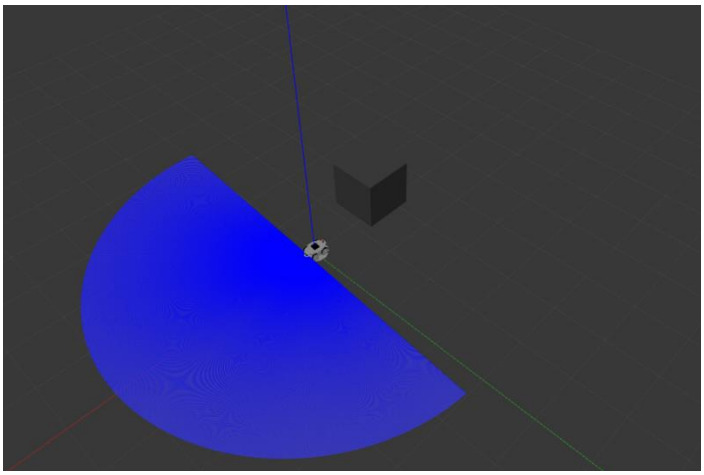

図 6-16　ビューに物体を追加

物体を削除する場合は、ビュー内で削除する物体をクリックしてから、キーボードの Delete キーを押してください。

② world 情報の保存・読み込み

シミュレーション上に追加した物体などの情報は、world ファイルとして保存することが可能です。メニューより「File」→「Save World As」をクリックして保存先を指定します。保存先

は必ずサンプルソース内の worlds ディレクトリにしてください。また、拡張子は.world にしてください。

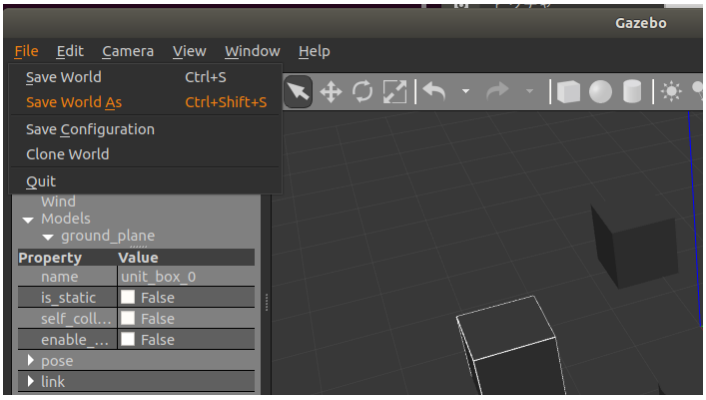

図 6-17　ビューを world に保存(できない場合はメカナムローバーを削除してみる)

なお、シミュレーション内にメカナムローバーが存在する場合、world ファイルが保存できない場合があります。保存ができない場合は、シミュレーション上からメカナムローバーを削除してください。

world ファイルを読み込んでシミュレーションを実行する場合は、下記の launch ファイルを使います。

```
roslaunch mecanumrover_samples vmecanumrover_withworld.launch
```

実行すると、サンプルソースに含まれるテスト用の world ファイルを読み込んで起動します。

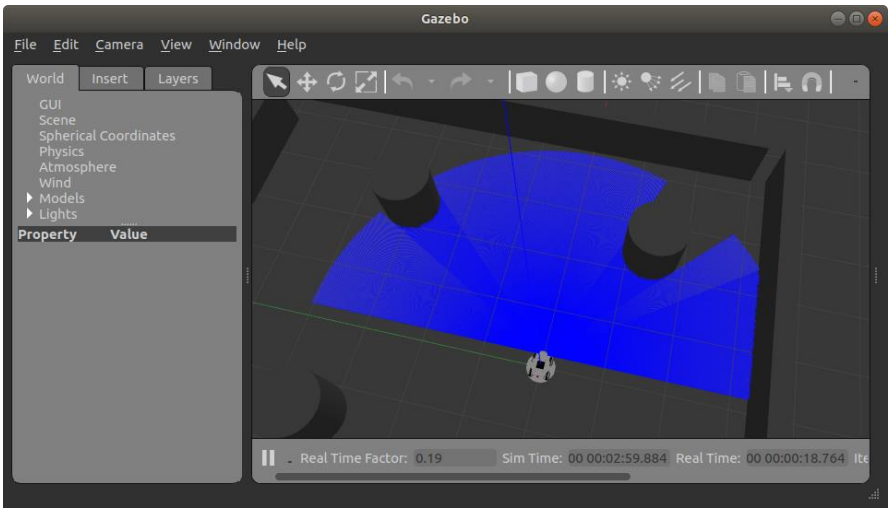

図 6-18　world ファイルを読み込んで起動

自分で作成して保存した world ファイルを読み込んで起動する場合は、下記のようにコマンドの末尾にオプションを追加してください。

```
roslaunch mecanumrover_samples vmecanumrover_withworld.launch
world:=(保存した world フィル名)
```

9 章以降で説明している SLAM や Navigation を行う場合、シミュレーション内に障害物が存在する必要があり、また Navigation では作成した地図と全く同一の地形で行う必要があります。これらを実行する場合は、サンプル及び自作の world ファイルを使用してください。

なお、各種操作方法に関するより詳しい説明は、下記の gazebo 公式ドキュメントをご参照ください。
http://gazebosim.org

③　オプションの設定

シミュレーション内のメカナムローバーは、2020 年 7 月現在提供している下記のオプション類について、一通り実装しています。

・全周囲バンパーオプション(前後左右 4 か所)
・LRF
・非常停止スイッチ
・ROS PC オプション(NUC PC)
・ワイヤレス充電オプション

オプション類の有効・無効化の設定は、urdf ディレクトリ内の vmegarover.xacro のファイルで行います。このファイルは xml 形式で記述されており、オプションに関するタグをコメントアウトすることで無効化します。

④　サスペンション機構のリフトアップ

メカナムローバーの実機には、ホイールやオプションのバンパーセンサ・非接触充電等にサスペンション機構が組み込まれています。シミュレーションにも同じ機構が組み込まれていますが、シミュレーションの仕様上、ミュレートの開始直後はうまく動作しないようになっています。そのため、別途専用ノードより、シミュレーション内時間で数秒間かけてサスペンション機構のリフトアップを行います。シミュレーションで制御を行う場合は、起動後シミュレーション内時間で 3 秒程度待ってから制御を開始してください。

⑤　シミュレーションと実機との差異

シミュレーション内のメカナムローバーは下記の要素について再現しております。

・無積載状態での並進・旋回速度
・各部位の寸法・衝突形状
・各部位の重量・イナーシャ

一方、下記の要素については明確に再現・検証しておりません。

・ホイールの摩擦係数
・積載時の挙動
・ダンパー機構の係数

シミュレーションを用いて開発を行う場合はこの点にご注意ください。また、シミュレータで再現している要素につきましても、必ずしも実機と同じ挙動を行うとは限らないため、ご注意ください。

6.5　メカナムローバー用メッセージの仕様

メカナムローバーは 1 つのメッセージをサブスクライブ（購読）し、2 つのメッセージをパブリッシュ（配信）しています。ここでは、各メッセージの仕様について解説します。

〔サブスクライブ〕

- /rover_twist

“/rover_twist”は、メカナムローバーへの移動指令を記載するメッセージです。メカナムローバーはこのメッセージをサブスクライブし、記載されている並進移動量[m/s]および旋回量[rad/s]の指令値に従って動作します。

表 6-1 に、“/rover_twist”の詳細を示します。メカナムローバーの座標系は、前方向を x 軸の正方向、右方向を y 軸の正方向にとる右手系で定義しています。メカナムローバーは全方向移動台車ですので、x 軸方向への移動および y 軸方向への移動と、z 軸周りでの旋回が行えます。

よって、“linear.x”, “linear_y”に並進移動量[m/s]を入力し、“angular.z”に旋回量[rad/s]を入力してパブリッシュすることで、メカナムローバーをコントロールすることができます。

表 6-1　/rover_twist の詳細

メッセージ名	/rover_tsist
型	geometry_msgs/Twist
内容	linear: 　x:　　//　x 軸方向の並進移動量指令値（float64） 　y:　　//　y 軸方向の並進移動量指令値（float64） 　z:　　//　不使用 angular: 　x:　　//　不使用 　y:　　//　不使用 　z:　　//　z 軸周りの旋回量指令値（float64）

〔パブリッシュ〕

- /rover_sensor

  “/rover_sensor”は、メカナムローバーのオプション品であるバンパーセンサの入力値を記録したメッセージです。

  表 6-2 に“/rover_sensor”の詳細を示します。ユーザは“/rover_sensor”をサブスクライブすることで、これらの値を利用した制御プログラムを作成することができます。

表 6-2　/rover_sensor の詳細

メッセージ名	/rover_sensor
型	std_msgs/Int16MultiArray
内容	layout: 　dim: [] 　data_offset: 0 data: [s0, s1]　//　s0　　-> VS-WRC052 の MU16_M_DI 　　//　s1　　-> VS-WRC051 の MU16_M_DI

- /rover_odo

  “/rover_odo”は、メカナムローバーの現在速度と旋回量を記録しているメッセージです。これらの値はエンコーダ値をもとに算出されています。本製品のサンプルパッケージに含まれるノード「pub_odom」は、“/rover_odo”をサブスクライブして、メカナムローバーの現在位置と姿勢を示すオドメトリ情報を計算して配信しています。

  表 6-3 に“/rover_odo”の詳細を示します。“/rover_odo”は、“rover_twist”と同じ、geometry_msgs/Twist 型のメッセージです。Twist 型は、移動量を表すためによく使われています。ROS には、データの種類に応じた沢山の型が用意されており、例えば「pub_odom」が配信するオドメトリ情報の型は nav_msgs/Odometry 型です。

表 6-3　/rover_odo の詳細

メッセージ名	/rover_odo
型	geometry_msgs/Twist
内容	linear: 　x:　　//　x 軸方向の並進移動量（float64） 　y:　　//　y 軸方向の並進移動量（float64） 　z:　　//　不使用 angular: 　x:　　//　不使用 　y:　　//　不使用 　z:　　//　z 軸周りの旋回量（float64）

7　ゲームパッド操作サンプル

ゲームパッド操作サンプルは、市販のゲームパッドを用いてメカナムローバーを走行させることができるサンプルプログラムです。本章では、その使用方法を説明します。

使用するゲームパッドは、アナログスティック入力機能を有する USB 接続のものを、お客様にてご用意ください。実機に付属しております VS-C3 は使用できません。また、ゲームパッドの種類によっては、デバイスドライバが非対応で動作しない可能性もございます。

① ゲームパッド入力パッケージ「Joy」のインストール

ROS でゲームパッドの入力を取得するためのパッケージ「Joy」をインストールします。端末を起動し、以下のコマンドを実行してください。

```
sudo apt install ros-ROS バージョン-joy ros-ROS バージョン-joystick-drivers
```

② ゲームパッドの接続とデバイスファイルのパスの確認

PC とゲームパッドは未接続の状態で端末を起動し、以下のコマンドを順に実行してください。

ゲームパッドのデバイスファイルが生成されるフォルダに移動します。

```
cd /dev/input
```

フォルダの中身を確認します。

```
ls
```

```
vstone@vstoneROSPC:~$ cd /dev/input/
vstone@vstoneROSPC:/dev/input$ ls
by-id    event0  event10  event12  event3  event5  event7  event9  mouse0
by-path  event1  event11  event2   event4  event6  event8  mice
```

図 7-1　ls の応答（接続前）

PC の USB ポートにゲームパッドを接続し、再度フォルダの中身を確認します。

```
ls
```

```
vstone@vstoneROSPC:/dev/input$ ls
by-id    event0  event10  event12  event2  event4  event6  event8  js0   mouse0
by-path  event1  event11  event13  event3  event5  event7  event9  mice
```

図 7-2　ls の応答（接続後）

図 7-1 と図 7-2 を比較すると、ゲームパッド接続後に“js0”が増えていることが分かります。これがゲームパッドのデバイスファイルです。よって、ファイルのパスは“/dev/input/js0”と

なります。デバイスファイル名は、機種やゲームパッドを接続するタイミングによって変化しますので、都度確認してください。

③　デバイスファイルパスをパラメータとして設定

手順①で確認したゲームパッドのデバイスファイルパスを、ゲームパッド操作サンプルの launch ファイル「joycon.launch」を使ってパラメータとして設定します。次の手順に沿って作業を実施してください。

端末を開き、gedit を使って joycon.launch を開きます。

```
gedit ~/catkin_ws/src/mecanumrover_samples/launch/joycon.launch
```

次の行を探してください。

```
<param name="dev" type="string" value="/dev/input/js0" />
```

launch ファイルでは、param 要素を用いることでパラメータを設定することができます。name 属性がパラメータ名、value 属性がパラメータ値です。“value=”以下を先ほど取得したゲームパッドのデバイスファイルパスに書き換えてください。書き換えが完了したら保存してください。

④　移動量の指令値の確認

メカナムローバーへの移動量指令値が正しく出力されるか確認します。**実機の場合は、安全のため、メカナムローバーと PC が接続されていない状態で作業を行ってください。**新しい端末で次のコマンドを実行し、joycon.launch を呼び出します。

```
roslaunch mecanumrover_samples joycon.launch
```

メカナムローバーへの移動指令が正確に出力されているか確認してみましょう。新しく端末をたち上げ、次のコマンドを実行してください。

```
rostopic echo /rover_twist
```

“rostopic echo メッセージ名”は、ノード間でやり取りされているメッセージを見ることができるコマンドです。コマンドと書きましたが、実際にはこれも ROS のノードのひとつとして実装されています。ここでは、メッセージ名に“/rover_twist”を指定して、メカナムローバーに送信されている移動速度[m/s]と、旋回量[rad/s]を確認します。

ゲームパッド操作サンプルにはセーフティーボタンが実装されており、デフォルトではゲームパッドの 3 番ボタンを押下している間のみ指令値が送信されます。ゲームパッドの 3 番ボタンを押下しながらアナログスティックを動かし、rostopic echo の応答を確認しましょう。

図 7-3　rostopic echo の応答（/rover_twist）

アナログスティックを動かすことで、図 7-3 のように linear.x および angular.z の値が変化すれば、指令値は正しく出力されています。linear は並進移動量を、angular は旋回量を表しており、x,y,z はそれぞれ座標軸です。

「何も値が出てこない場合」や「片方の値しか変化しない場合」は、手順④の操作方法のカスタマイズが必要です。あるいは、値は両方とも変化したが、ボタンやアナログスティックの割り当てが不自然で操作しづらいときなども、操作方法のカスタマイズを行ってください。

指令値が正しく出力されていて、操作のしやすさにも問題がない場合は、手順⑤に進んでください。

⑤　操作方法、パラメータのカスタマイズ

joycon.launch を編集することで、操作方法や最高速度などを変更することができます。次のコマンドで joycon.launch を開いてください。

```
gedit ~/catkin_ws/src/mecanumrover_samples/launch/joycon.launch
```

以下に示す joycon の node 要素内にある各 param 要素の value 属性値を編集します。

```
<!-- joycon -->
<node pkg=”mecanumrover_samples” type=”joycon” name=”joycon”
respawn=”true” >
  <param name=”axis_linear” value=”1” />
  <param name=”axis_angular” value=“0” />
  <param name=”scale_linear” value=”0.6” />
  <param name=”scale_angular” value=”0.8” />
  <param name=”safety_button” value=”2” />
```

各パラメータの意味を次表に示します。

表 7-1　joycon のパラメータ一覧

パラメータ名	説明	初期値
axis_linear_x	前後の移動速度[m/s]を決めるアナログスティックの番号 /joy メッセージの axes 配列の要素番号から選択	1
axis_linear_y	左右の移動速度[m/s]を決めるアナログスティックの番号 /joy メッセージの axes 配列の要素番号から選択	0
axis_angular	旋回量[rad/s]を決めるアナログスティックの番号 /joy メッセージの axes 配列の要素番号から選択	2
scale_linear	移動量のゲイン（1.3 までに留めること） 移動量＝scale_linear×アナログスティック入力値	0.6
scale_angular	旋回量のゲイン（6.0 程度までに留めること） 旋回量=scale_angular×アナログスティック入力値	0.8
safety_button	セーフティーボタンの番号 /joy メッセージの button 配列の要素番号から選択	2

“axis_linear_x/y”、“axis_angular”、“safety_button”を編集することで操作方法を変更できます。**実機の場合は、安全のため、メカナムローバーと PC が接続されていない状態で**以下のコマンドを実行してください。

```
roslaunch mecanumrover_samples joycon.launch
```

ゲームパッドからの入力値を確認します。

```
rostopic echo /joy
```

```
---
header:
  seq: 55
  stamp:
    secs: 1521521432
    nsecs:   9187362
  frame_id: ''
axes: [0.07760214805603027, 0.3597537875175476, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
```

図 7-4　rostopic echo の応答（/joy）

図 7-4 に、ゲームパッドの 3 番ボタンを押下しアナログスティックを倒したときの“rostopic echo /joy”の応答を示します。アナログスティックの入力値は axes 配列に、ボタンの入力値は buttons 配列に格納されます。どの要素にどのスティック、ボタンの値が格納されるかはゲームパッドによって異なります。ゲームパッドを操作して適当なパラメータを設定してください。

“scale_linear”、“scale_angular”を編集することで、ゲームパッドの操作量に対する応答を変化させることができます。メカナムローバーが発揮できる速度には限界がありますので、大きな値を設定すると操作性が失われ、大変危険です。表 7-1 を参考に慎重に変更を行ってください。

⑥ ゲームパッド操作サンプルの実行

手順②、③で書き換えた launch ファイルを呼び出し、メカナムローバーをゲームパッドで操作します。**実機の場合は、**メカナムローバーが壁などと衝突しないよう十分なスペースを確保してください。また、PC はメカナムローバーにしっかりと固定し、落下などしないようご注意ください。ケーブルが絡まったりすることがないように、ケーブルの状態を確認しつつ実施してください。

実機の場合は、メカナムローバーの電源が OFF になっていることを確認し、6.3 項に従って PC とメカナムローバーを接続します。接続が正常に行われていることを確認したら、新しい端末で次のコマンドを実行し、joycon.launch を呼び出します。

シミュレーションを利用する場合は、6.4 項に従ってシミュレーションを起動してください。

```
roslaunch mecanumrover_samples joycon.launch
```

実機の場合は、**ゲームパッドを操作していない状態でメカナムローバーの電源を入れます。**ゲームパッドをゆっくりと操作し、メカナムローバーが走行することを確認します。アナログスティックを急激に操作すると危険ですのでおやめください。

シミュレーションの場合は、電源投入等の作業は不要です。シミュレータを起動したら、サスペンションのリフトアップを待ってから、前述の操作設定を参考にゲームパッドで操縦してください。

8　マウス（タッチパッド）操作サンプル

マウス（タッチパッド）操作サンプルは、マウスまたはタッチパッドを使い、メカナムローバーを走行させることができるサンプルです。本章ではその使用方法を説明します。

① mouse_teleope パッケージのインストール

マウス（タッチパッド）操作サンプルの実行には mouse_teleope パッケージのインストールが必要です。端末を立ち上げ、以下のコマンドを実行してください。

```
sudo apt install ros-ROS バージョン-mouse-teleope
```

② マウス（タッチパッド）操作サンプルの実行

launch ファイルを用いてサンプルを起動します。実機の場合は、メカナムローバーが壁などと衝突しないよう十分なスペースを確保してください。また、PC はメカナムローバーにしっかりと固定し、落下などしないようご注意ください。ケーブルが絡まったりすることがないように、ケーブルの状態を確認しつつ実施してください。

実機の場合は、安全のため、メカナムローバーと PC が接続されていない状態で mousectrl.launch を呼び出します。roscore を起動していない場合は別端末にて先に起動しておいてください。

```
roslaunch mecanumrover_samples mousectrl.launch
```

mousectrl.launch を呼び出すと、図 8-1 のようなウィンドウが画面に現れます。白いエリアをクリックし、上下方向にドラッグすると移動量 v が、左右方向にドラッグすると旋回量 w が出力されます。

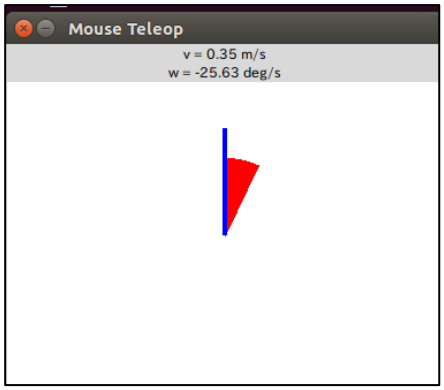

図 8-1　Mouse Tekeop の GUI

実機の場合は、**メカナムローバーの電源が OFF になっていることを確認し**、6.3 項に従って PC とメカナムローバーを接続します。その後メカナムローバーの電源を ON にしてマウスを操作

すると、メカナムローバーが走行します。シミュレーションの場合は、6.4 項に従ってシミュレータを起動してください。起動後、サスペンションのリフトアップを待ってから、マウスで操作してください。

9　SLAM（gmapping）サンプル

SLAM(Simultaneous Localization and Mapping)は、ロボットの自己位置推定と環境地図作成を同時に行う手法です。ロボット掃除機などの自律移動ロボットでよく用いられており、現在も盛んに研究されている高度な技術です。

ROS には、この SLAM 手法を実装したパッケージが存在しており、誰でも簡単に SLAM を利用することが可能となっています。ROS で SLAM を実装したパッケージはいくつか存在しますが、ここでは「gmapping」を利用して、SLAM を実行してみましょう。

SLAM（gmapping）サンプルは、メカナムローバー前部に取り付けられた LRF で取得した情報をもとに、自己位置の推定と周囲の環境地図の作成を行います。メカナムローバーの移動は手動で行う必要がありますので、ゲームパッド操作サンプルまたはマウス（タッチパッド）操作サンプルを一緒に使用します。

LRF(Laser Rangefinder）は、周囲に存在する物体の位置を検出することができるセンサです。レーザー光を発射し、その反射を見て物体の有無を確認しています。視野が広く、計測精度が高いことから、自律移動台車ロボットなどでよく使用されています。

「gmapping」は ROS でメジャーな SLAM パッケージのひとつです。LRF のデータと、車輪の回転数から移動量を計測するオドメトリの情報を用いて SLAM を行います。「gmapping」を用いて作成した地図の例を図 9-1 に示します。

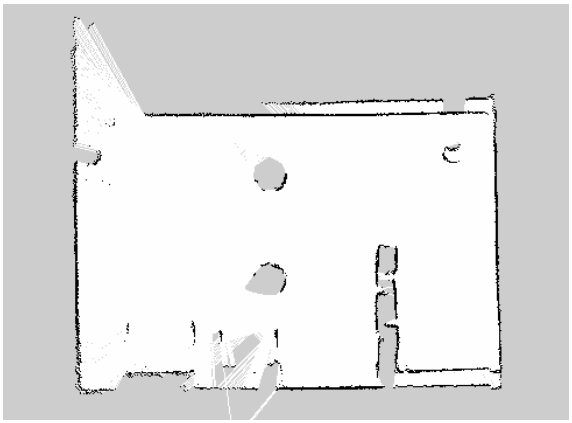

図 9-1　gmapping の地図作例

図中の黒い部分は障害物が存在する通行不可能な箇所、白い領域は通行可能な場所、灰色は未知領域です。地図はグリッド状になっており、各グリッドには障害物が存在する確率（占有確率）を示す 0～100 の値もしくは、未知領域を示す-1 の値が与えられます。この占有確率を用いた地図の表現方法は、ROS の 2D マップ表現方法として標準化されています。gmapping は絶対に障害物が存在しない 0 もしくは、絶対に障害物が存在する 100 の 2 種の値しか出力しません。

9.1 SLAM（gmapping）サンプルの実行

SLAM（gmapping）サンプルを実行する手順を説明します。以下の手順に従って作業を行ってください。

① gmapping のインストール

slam-gmapping パッケージをインストールします。次のコマンドを実行してください。

```
sudo apt install ros-ROS バージョン-slam-gmapping
```

② メカナムローバーのセットアップ

実機の場合は、**メカナムローバーの電源が OFF になっていることを確認し**、6.3 項に従いメカナムローバーと ROS を接続してください。シミュレーションの場合は、6.4 項に従いシミュレータを起動しますが、障害物が何もないと地図が作成されないため、必ず障害物込みの launch ファイルを実行するか、手動で障害物を追加してください（手動で障害物を追加した場合、今後同じ障害物配置を再現するため、必ず地形を保存してください）。

続いて、メカナムローバーを手動で操作できるようにします。ゲームパッドを使用する場合は 7 章の、マウスまたはタッチパッドを使用する場合は 8 章の手順に従い、各サンプルを使用して、メカナムローバーを操作できることを確認してください。

③ launch ファイルの修正

新しい端末で次のコマンドを実行し、gmapping.launch を開いてください。

```
gedit ~/catkin_ws/src/mecanumrover_samples/launch/gmapping.launch
```

修正内容は実機とシミュレーションで異なります。それぞれの動作環境に応じて書き換えてください。

・実機の場合

次の項目を確認し、default 属性値を手順②で確認した LRF のデバイスファイルパスに変更して、保存してください。

```
<!-- LRF のデバイスファイルパス  -->
<arg name="port_urg" default="/dev/ttyACM0" /> <!-- LRF のデバイスファイルパスを設定してください  -->
```

・シミュレーションの場合

下記の項目の赤字部分をそれぞれコメントアウトしてください。コメントアウトは、「<!--」と「-->」で括ることでできます。複数のコメントアウトを入れ子上に記述できない(例:「<!- - <!-- ~~~ --> -->」のような記述は不可能)ため注意して下さい。

```
<!-- LRF のデバイスドライバノード -->
  <node pkg="urg_node" type="urg_node" name="urg_node">
      <param name="serial_port" value="$(arg port_urg)" />
      <param name="angle_min" value="-1.57" /> <!-- 視野角の指定（最小値） -->
      <param name="angle_max" value="1.57" />  <!-- 視野角の指定（最大値） -->
  </node>
```

④ SLAM の実施

新しい端末で次のコマンドを実行し、SLAM（gmappig）サンプルを起動します。

```
roslaunch mecanumrover_samples gmapping.launch
```

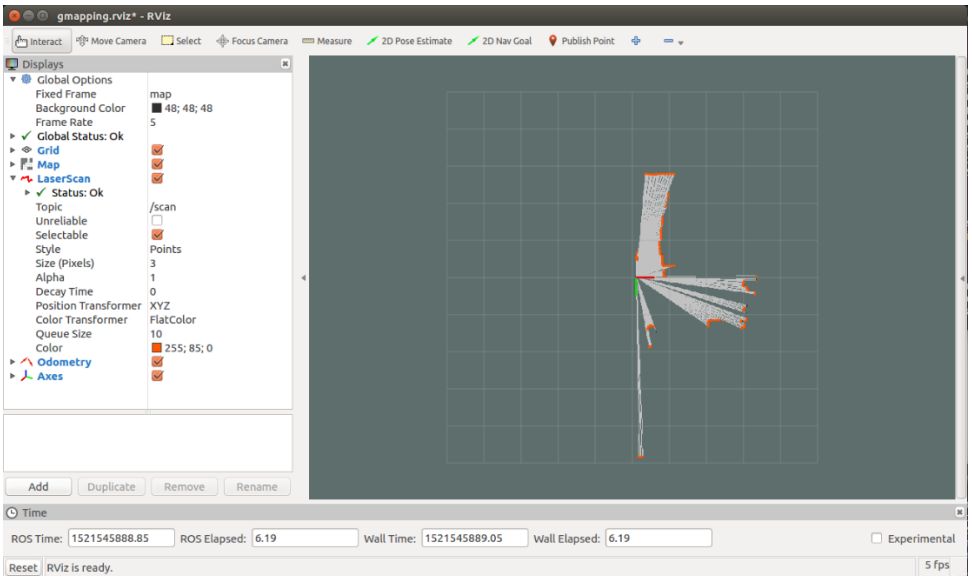

図 9-2　Rviz による gmapping の表示

サンプルが起動すると、図 9-2 に示すように Rviz が起動します。Rviz 上には LRF が捉えている障害物と、生成される地図がリアルタイムで表示されます。画面が表示されなかったり、LRF のデータが表示されなかったりした場合は、起動中に何らかのエラーが発生した可能性があります。一度プログラムを終了し、再度実行しなおしてください。

地図が生成され始めたら、メカナムローバーをゆっくりと移動させていきましょう。移動した範囲の地図が徐々に作成されていくはずです。急発進や急停止、急旋回などを行うと地図が乱れやすいので注意してください。

9.2 map の保存

作成した map を保存する方法を説明します。map の保存には map_server パッケージの map_saver ノードを使用します。次のコマンドを実行することで、ホームフォルダに地図の画像ファイルとデータファイルが保存されます。ファイル名は適宜設定してください。

```
rosrun map_server map_saver -f ファイル名
```

map_server または map_saver が見つからないというエラーが出てマップが保存できない場合、必要なパッケージが不足しています。11.1 節を参考に、navigation スタックをインストールし、map_server をインストールしてください。

10 SLAM（cartographer）サンプル

「cartographer」は 2016 年に発表された比較的新しい SLAM 手法です。ここでは cartographer を ROS に対応させた「cartographer_ros」を用いて SLAM を行ってみましょう。

なお、cartgrapher サンプルについては、シミュレーション機能では対応・検証しておりませんのでご了承ください。

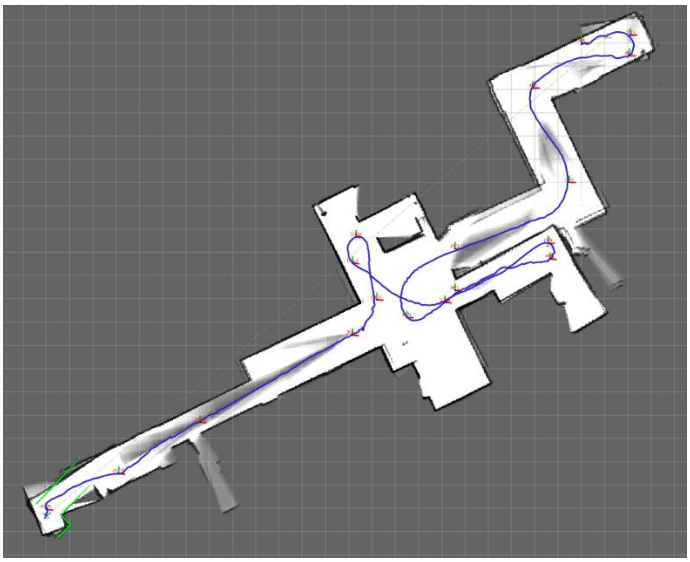

図 10-1　cartographer の地図作例

cartographer を用いて作成した地図の例を図 10-1 に示します。cartographer で作成される地図も ROS の標準化された表現方法に準拠していますので、gmapping で作成したものと同様に、壁などの障害物が黒で、通行可能な領域が白で表されています。gmapping との差としては、占有確率の出力値が異なることが挙げられます。gmapping では 0 か 100 かで出力されていた占有確率が、cartographer では 0～100 までリニアに出力されます。

10.1 SLAM（cartographer）サンプルの実行

SLAM（cartographer）サンプルを実行する手順を説明します。以下の手順に従って作業を行ってください。なお、インストール方法のほか、cartographer_ros に関する様々なドキュメントは、次のサイトで確認することができます。

Cartographer ROS - https://google-cartographer-ros.readthedocs.io/en/latest/

① cartographer_ros のインストール

cartographer_ros パッケージと、その依存パッケージをソースからビルドしてインストールします。cartographer_ros は、catkin の一般的なビルドコマンド“catkin_make”ではなく、“catkin_make_isolated”コマンドを使用してビルドします。“catkin_make”と“catkin_make_isolated”では、ビルド後に生成されるフォルダ構造が異なります。混乱とエラーを防ぐために、新たなワークスペースを作成して作業を行いましょう。

ワークスペース管理ツール wstool とビルド高速化ツール ninja を導入します。

```
sudo apt update
```

```
sudo apt install -y python-wstool python-rosdep ninja-build
```

次のコマンドを順に実行し、新しいワークスペースを作成しましょう。

```
mkdir -p ~/catkin_ws_isolated
```

```
cd ~/catkin_ws_isolated
```

```
wstool init src
```

Github から cartographer_ros のインストールファイルをマージします。

```
wstool merge -t src
https://raw.githubusercontent.com/googlecartographer/cartographer_ros/master/cartographer_ros.rosinstall
```

```
wstool update -t src
```

proto3 をインストールします。

```
src/cartographer/scripts/install_proto3.sh
```

依存関係をインストールします。

```
rosdep update
```

```
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

ビルドとインストールを実行します。

```
catkin_make_isolated --install --use-ninja
```

オーバーレイの設定を行います。

```
echo "source /home/ユーザ名/catkin_ws_isolated/install_isolated/setup.bash"
>> ~/.bashrc
```

```
source ~/.bashrc
```

以上で、cartographer_ros のインストールは完了です。

② メカナムローバーのセットアップ

メカナムローバーの電源が OFF になっていることを確認し、6.3 項に従いメカナムローバーと ROS を接続してください。

続いて、メカナムローバーを手動で操作できるようにします。ゲームパッドを使用する場合は 7 章の、マウスまたはタッチパッドを使用する場合は 8 章の手順に従い、各サンプルを使用して、メカナムローバーを操作できることを確認してください。

③ メカナムローバーのセットアップ

新しい端末で次のコマンドを実行し、cartographer.launch を開いてください。

```
gedit ~/catkin_ws/src/mecanumrover_samples/launch/cartographer.launch
```

次の項目を確認し、value 属性値を手順②で確認した LRF のデバイスファイルパスに変更して、保存してください。

```
<param name="serial_port" value="/dev/ttyACM0" />
```

④ SLAM の実施

新しい端末で次のコマンドを実行し SLAM（cartographer）サンプルを起動します。

```
roslaunch mecanumrover_samples cartographer.launch
```

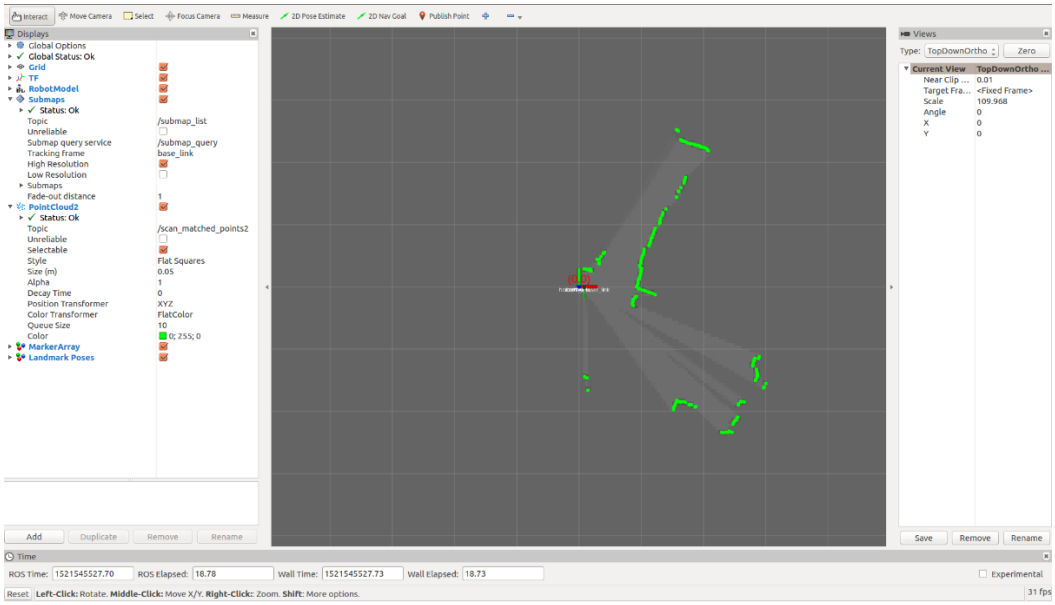

図 10-2　Rviz による cartographer の表示

サンプルが起動すると図 10-2 に示すように Rviz が起動します。Rviz 上には LRF が捉えている障害物と、生成される地図がリアルタイムで表示されます。画面が表示されなかったり、LRF のデータが表示されなかったりした場合は、起動中に何らかのエラーが発生した可能性があります。一度プログラムを終了し、再度実行しなおしてください。

地図が生成され始めたら、メカナムローバーをゆっくりと移動させていきましょう。移動した範囲の地図が徐々に作成されていくはずです。急発進や急停止、急旋回などを行うと地図が乱れやすいので注意してください。

11 navigation サンプル

ロボットの自律移動を行うためには、自己位置推定や移動経路のプランニングといった技術が必要です。ROS では navigation スタックという形でこれらの機能が提供されています。スタックとは、複数のパッケージを束ねたパッケージ群のことです。ここでは、SLAM（gmapping）サンプルを用いて作成した地図を使って、メカナムローバーを目標地点まで自律的に移動させてみましょう。

11.1 navigation スタックのインストール

navigation スタックをインストールします。navigation スタックに属する各パッケージは、以下のコマンドを実行することで一括でインストールすることができます。

```
sudo apt install ros-ROS バージョン-navigation
```

11.2 地図の作成と launch ファイルの編集

まず、ロボットが移動する環境の地図を作成します。自律移動を行うためには、できるだけ正確な地図が必要です。また、launch ファイルを編集し、作成した地図が読み込まれるようにします。

① 地図の作成

9.1 項の手順に従い gmapping を実行し、navigation で移動させたい領域の地図を作成してください。本来障害物が無いはずの場所に障害物があると判定された場合は、再び同じ個所を走行することで地図が正しく修正されることがあります。

② 地図の保存

9.2 項の手順に従い、map_saver を用いて地図を保存してください。地図が保存されていることを確認したら、SLAM（gmapping）サンプルは終了させてください。

③ 地図ファイルの移動

保存した pgm ファイルと yaml ファイルを以下のアドレスに移動させてください。

```
~/catkin_ws/src/mecanumrover_samples/map
```

④ launch ファイルの編集

mecanumrover_move_base_dwa.launch を編集し、作成した地図が読み込まれるようにします。以下のコマンドで launch ファイルを開いてください。

```
gedit
~/catkin_ws/src/mecanumrover_samples/launch/mecanumrover_move_base_dwa.launch
```

下記の行を探し、“ファイル名.yaml”を、先ほど保存，移動させた地図ファイルのファイル名に合わせて設定します。

```
<arg name=”map_file” default=”$(find mecanumrover_samples)/map/ファイル
```

また、シミュレーションの場合に限り、下記の赤字部分をコメントアウトしてください。

```
  <node name="horizontal_laser" pkg="urg_node"
    type="urg_node" >
    <param name="serial_port" value="/dev/ttyACM0" />  <!-- デバイスドライバファイルのパス  -->
    <param name="frame_id" value="horizontal_laser_link" />
    <param name="angle_min" value="-1.57" />  <!-- LRF の有効検出角度（下限）  -->
    <param name="angle_max" value="1.57" />  <!-- LRF の有効検出角度（上限）  -->
  </node>
```

11.3 navigation サンプルの実行

それでは navigation サンプルを実行して、メカナムローバーに自律移動をさせてみましょう。なお、環境によっては上手く自律移動できないことがあります。その場合は、各機能のパラメータを調整いただくことで、改善する可能性があります。サンプルプログラムではオドメトリの影響が小さくなるようパラメータを設定しています。

平面で、オドメトリの誤差原因となるタイヤの滑りが少なく、分かれ道や柱の凹凸など、分かりやすい特徴が存在する通路といった環境でお試しください。LRF では捉えられない高さに障害物があったり、イスやテーブルの脚といった小さな障害物が多く存在する環境では正常に動作させることが難しくなります。

以下の手順に従って作業を行ってください。11.1 項を実施した後、実機の場合、メカナムローバーと ROS との接続を解除している場合は、6.3 項に基づいて接続作業を行ってください。シミュレーションの場合、シミュレータを終了している場合は 6.4 項に基づいてシミュレータを起動してください。この時、必ず地図ファイル作成時と同じ地形を読み込ませてください。

① launch ファイルの呼び出し

メカナムローバーと ROS が接続された状態で、以下のコマンドを実行して launch ファイルを呼び出します。

```
roslaunch mecanumrover_samples mecanumrover_move_base_dwa.launch
```

図 11-1 に示すように、Rviz 上に 11.1 項で作成した地図と現在の LRF が捉えた物体情報が表示されることを確認してください。

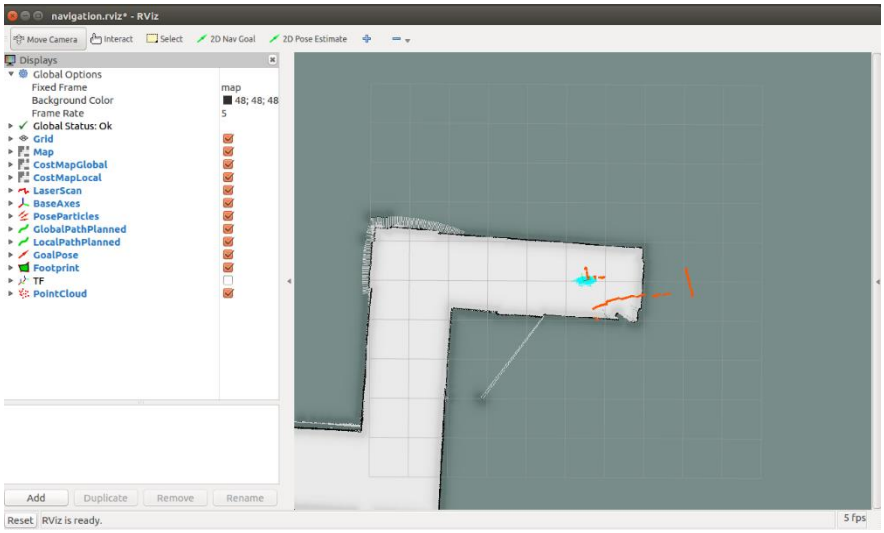

図 11-1　navigation サンプルの Rviz 画面

② 初期位置の設定

navigation サンプルが起動出来たら、メカナムローバーのおおよその初期位置を教えてやる必要があります。画面上部の「2D Pose Estimate」ボタンをクリックしてから、図 11-2 に示すように、メカナムローバーの実際の初期位置をクリックし、ドラッグして方向を決めます。

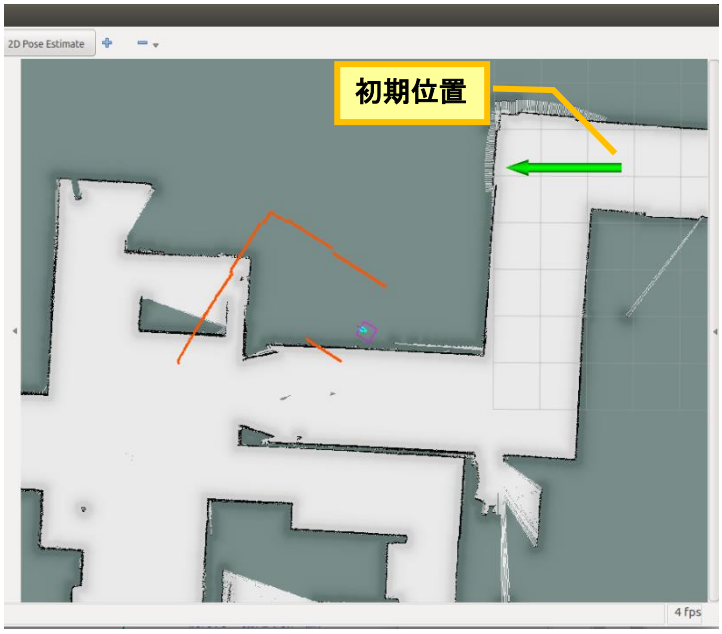

図 11-2　メカナムローバー初期位置の設定

図 11-3 に示すように、map の壁や障害物と、LRF の検出結果がおおよそ一致するようになるまで調整を行ってください。

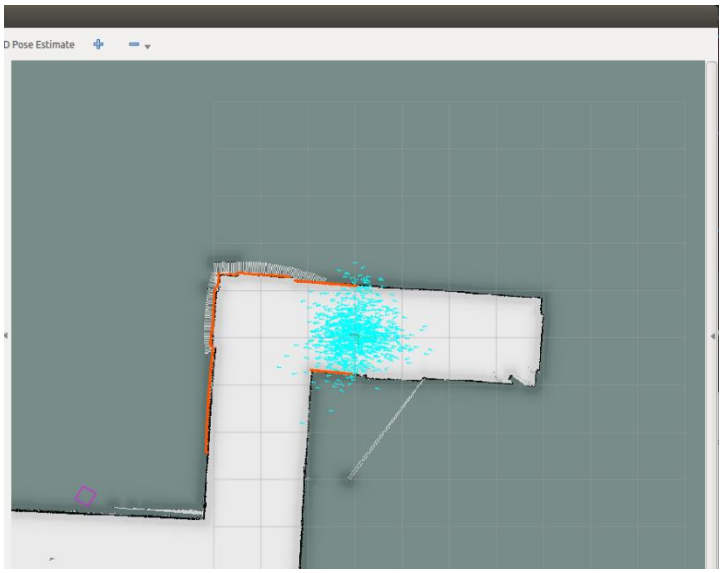

図 11-3　初期位置の設定が完了した状態

このとき、指定したメカナムローバーの位置周辺に青い小さな点がいくつも散らばっていることが確認できます。これらは、navigation スタック内で自己位置推定を担うノード「amcl」が計算する、メカナムローバーの推定自己位置です。動作開始前は自己位置が定まっていないため、青い点は大きく散らばっています。

③　目標位置の設定

初期位置の設定を完了したら、いよいよメカナムローバーの移動目標地点を設定します。

なお目標地点を設定すると、メカナムローバーは即座に走行を開始しますので注意してください。

画面上部の「2D Nav Goal」ボタンをクリックしてから、図 11-4 に示すように、目標位置をクリックし、ドラッグして方向を決めてください。

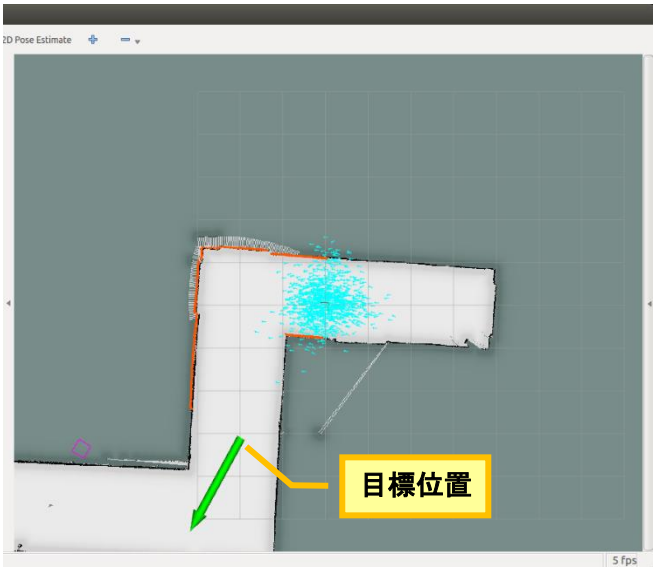

図 11-4　目標位置の設定

目標位置を設定すると、メカナムローバーは即座に走行を開始します。走行中の Rviz の様子を図 11-5 に示します。計画された移動経路が緑線で、目標位置姿勢が赤矢印で示されています。また、検出されている障害物の周りには色が付いた領域が存在します。これらは、観測結果をもとに数値化された衝突危険性を表すコストマップです。

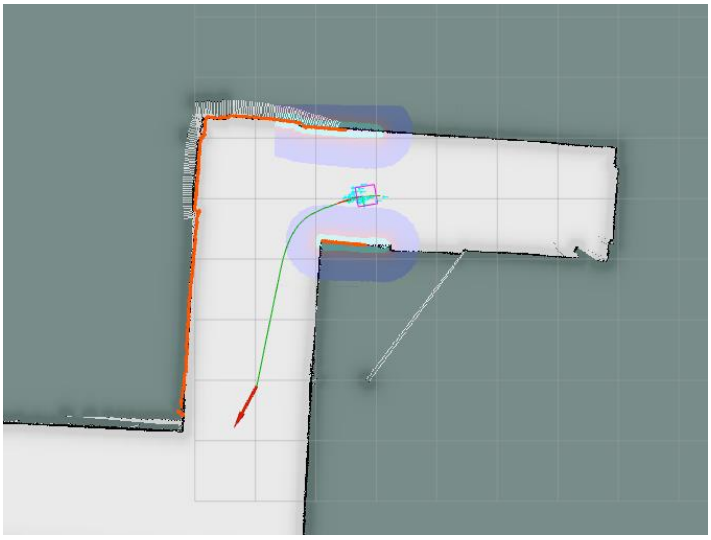

図 11-5　navigation による走行の様子

④　走行の終了

メカナムローバーは目標位置に到達すると、自動的に停止します。そこから新たに目標位置を設定することも可能です。

12 動作確認バージョンについて

メカナムローバーの各サンプルが使用しているパッケージについては、以下のバージョンで動作確認を行っております。“apt”コマンドを用いてバージョン指定のオプションを付けずにバイナリを導入した場合、動作確認を行っていないバージョンが導入されシステムが正常に動作しない可能性がございます。また、Github からソースコードを取得してビルドする導入方法でも同様です。その際は、バージョン指定のオプションを用いて弊社にて動作確認を行っているバージョンのパッケージを導入してください。

表 12-1 動作確認パッケージのバージョン

パッケージ名	バージョン
rosserial	0.7.7
mouse-teleop	0.2.6
gmapping	1.3.10
cartographer	1.0.0
navigation	1.14.4

商品に関するお問い合わせ
TEL: 06-4808-8701　　FAX: 06-4808-8702　　E-mail: infodesk@vstone.co.jp
受付時間　：9:00～18:00（土日祝日は除く）
ヴイストン株式会社
〒555－0012　大阪市西淀川区御幣島 2-15-28
www.vstone.co.jp