

# Autoware ユーザーズマニュアル

2015/AUG/03

名古屋大学

## 内容

はじめに .....	4
概要 .....	4
用語 .....	4
関連文書 .....	6
全体構成 .....	7
構成 .....	7
主な機能 .....	8
環境構築の手順 .....	9
Linux .....	9
ROS .....	10
OpenCV .....	10
Qt .....	10
CUDA .....	11
FlyCapture2 .....	12
Autoware .....	13
AutowareRider .....	13
canlib .....	14
SSH の公開鍵の作成 .....	14
使用手順 .....	15
センサデータの取得 .....	15
自動運転 .....	15
AutowareRider .....	15
概要 .....	15
起動方法 .....	16
経路データ生成アプリケーションの使用方法 .....	17
ROS PC への経路データ転送手順 .....	17
CAN データ収集アプリケーションの使用方法 .....	17
ROS PC への CAN データ転送手順 .....	18
Launch ファイルの起動方法 .....	19
各機能の説明 .....	19
ROS .....	19
認知（物体検出，位置推定） .....	19
判断（レーン走行，交差点） .....	19
操作 .....	19
データ .....	19
センサ .....	21
非 ROS モジュールとの通信 .....	21

ユーティリティ .....	22
Runtime Manager .....	22
概要 .....	22
Quick Start タブ .....	23
Map タブ .....	27
Sensing タブ .....	29
Computing タブ .....	33
Interface タブ .....	38
Database タブ .....	40
Simulation タブ .....	43
Status タブ .....	45
ユーザインタフェース .....	46
概要 .....	46
AutowareRider .....	46
AutowareRoute .....	50
車の制御 .....	52
一般 .....	53
ZMP .....	53

# 先頭が「#」で始まる行は、コメントです。

## はじめに

### 概要

この文書は、Linux と ROS(Robot OS)をベースとした、自動運転を実現するためのオープンソースのソフトウェアパッケージ「Autoware」の{ユーザース|デベロッパーズ}マニュアルです。

(ユーザースマニュアルでは、) Autoware と、各種センサ機器もしくはデータを使用して、自動運転もしくはその一部の機能を動作させる手順について記述しています。

(デベロッパーズマニュアルでは、) Autoware に独自の機能を追加するために必要な開発手順、その助けとなる情報について記述しています。

### 用語

# 自動運転に関する用語も、統一したいので追加する。

- ROS (Robot Operating System)

ロボットソフトウェア開発のためのソフトウェアフレームワーク。ハードウェア抽象化や低レベルデバイス制御、よく使われる機能の実装、プロセス間通信、パッケージ管理などの機能を提供する。

- パッケージ (Package)

ROS を形成するソフトウェアの単位。ノードやライブラリ、環境設定ファイルなどを含む。

- ノード (Node)

単一の機能を提供するプロセス。

- メッセージ (Message)

ノード同士が通信する際のデータ構造。

- トピック (Topic)

メッセージを送受信する先。メッセージの送信を「Publish」、受信を「Subscribe」と呼ぶ。

- OpenCV (Open source Computer Vision library)

コンピュータビジョンを扱うための画像処理ライブラリ。

- Qt

アプリケーション・ユーザ・インタフェースのフレームワーク。

- CUDA (Compute Unified Device Architecture)

NVIDIA 社が提供する、GPU を使った汎用計算プラットフォームとプログラミングモデル。

- FlyCapture SDK

PointGrey 社のカメラを制御するための SDK。

- FOT (Field Operation Test)

実道実験。

- GNSS (Global Navigation Satellite System)

衛星測位システム。

- LIDAR (Light Detection and Ranging または Laser Imaging Detection and Ranging)

レーザー照射を利用して距離などを計測する装置。

- DPM (Deformable Part Model)

物体検出手法。

- KF (Kalman Filter)

過去の観測値をもとに将来の状態を推定する手法。

- NDT (Normal Distributions Transform)

位置推定手法。

- キャリブレーション

カメラに投影された点と 3 次元空間中の位置を合わせるための、カメラのパラメータを求める処理。

- センサ・フュージョン

複数のセンサ情報を組合せて、位置や姿勢をより正確に算出するなど、高度な認識機能を実現する手法。

- TF (TransForm?)

ROS の座標変換ライブラリ？

- オドメトリ (Odometry)

車輪の回転角と回転角速度を積算して位置を推定する手法。

- SLAM (Simultaneous Localization and Mapping)

自己位置推定と環境地図作成を同時に行うこと。

- CAN (Controller Area Network)

自動車等の内部で相互接続された機器間のデータ転送に使用される規格。

- IMU (Inertial Measurement Unit)

慣性計測装置。角速度や加速度を計測する装置。

- DMI (Distance Measuring Instrument)

走行距離計。

-

## 関連文書

# 文書ではなく URL になっていますが...

- Autoware  
<http://www.pdsl.jp/fot/autoware/>
  - ROS  
<http://www.ros.org/>
  - OpenCV  
<http://opencv.org/>  
<http://opencv.jp/>
  - Qt  
<http://www.qt.io/>  
<http://qt-users.jp/>
  - CUDA  
[http://www.nvidia.com/object/cuda\\_home\\_new.html](http://www.nvidia.com/object/cuda_home_new.html)  
<http://www.nvidia.co.jp/object/cuda-jp.html>
  - FlyCapture SDK  
<http://www.ptgrey.com/flycapture-sdk>
  -
-

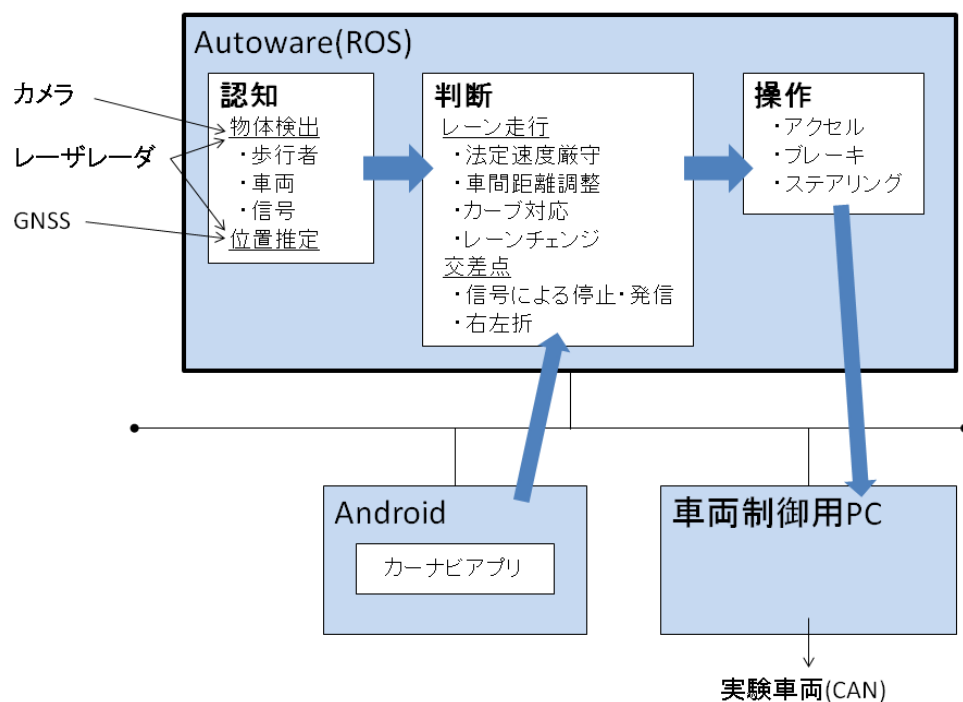
## 全体構成

# Autoware の PC + 各種センサ機器 の図と説明を書く。

# Autoware の中は、加藤先生の仕様書などを参考に。

# <http://www.pdsl.jp/fot/autoware/>

Autoware は、Linux と ROS をベースとした、自動運転を実現するためのオープンソースのソフトウェアパッケージです。レーザレーダ、カメラ、GNSS などの環境センサを使用して、自車位置や周囲物体を認識しながら、カーナビから与えられたルート上を自立走行することができます。



## 構成

Autoware による自動運転の機能は、自己位置推定や周囲物体の検出などを行う「認知」、レーンや交差点での走行・停止の「判断」、実際の車両の「操作」、の3つに分けられます。

- `ros/src/computing/perception/`  
認知
- `ros/src/computing/planning/`  
判断
- `ros/src/computing/control/`  
操作

- `ros/src/data/`  
3次元地図などのデータの読み込み(DB、ファイル)
- `ros/src/sensing/`  
各種センサドライバ、キャリブレーション、フュージョンなど
- `ros/src/socket/`  
スマートフォン用アプリケーションとのインタフェース
- `ros/src/util/`  
Runtime Manager、サンプルデータ、擬似ドライバなど
- `ui/tablet/`  
スマートフォン用アプリケーション
- `vehicle/`  
車両の制御、情報取得など
- 

## 主な機能

Autoware には以下のような機能があります。

また、これらを実施するためのユーザインタフェース(Runtime Manager)も用意されています。

- 自己位置推定  
3次元点群地図と3次元LIDARデータを入力として、NDTアルゴリズムをベースとしたスキャンマッチングを行うことで、自車位置を10cm程度の誤差で推定することができます。
- 3次元地図生成  
SLAM技術を用いて、3次元地図をリアルタイムに生成することができます。  
生成した3次元地図を、既存の3次元地図に追加することも可能です。この機能により、3次元地図のオンライン更新も実現できます。  
3次元地図から地物データを抽出することで、ベクタ形式の3次元地図を生成することもできます。
- 信号機検出  
自己位置推定の結果と高精度3次元地図から、信号機の位置を正確に算出し、信号機の3次元位置をセンサフュージョンによってカメラ画像上に射影します。そこから画像処理によって色判別することで、信号機を検出することができます。
- 物体検出  
カメラ画像を入力として、DPMアルゴリズムによる画像認識を行うことで、車両や歩行者を検出することができます。



KF を利用してトラッキングを行うことも可能です。トラッキング機能を導入すると、個々の物体を追跡でき、かつ誤認識を削減できます。

また、3 次元 LIDAR データをフュージョンすることで、検出した物体までの距離も算出できます。

- 経路生成

自動運転の経路は、スマートフォンのカーナビアプリケーション (MapFan を使用した経路データ生成アプリケーション) から入力できます。経路には適切な速度情報も含まれ、その速度を目安に自立走行します。

- 経路追従

生成した経路に 1m 間隔の目印 (way point) を設定し、その目印を追っていくことで経路追従を行います。カーブでは近くの way point、直線では遠くの way point を参照することで、自立走行を安定化しています。

経路から逸脱した場合は、近傍の way point を目指して経路に戻ります。

- 

---

## 環境構築の手順

PC に、以下の手順で、Linux、ROS、Autowareなどをインストールする手順を示します。

CUDA と FlyCapture SDK は、必須ではありません。

NVIDIA 社のグラフィックボードに搭載された GPU を使って計算を行う場合は、CUDA が必要です。また、PointGrey 社のカメラを使用する場合は、FlyCapture SDK が必要です。

### Linux

現時点で、Autoware が対応している Linux ディストリビューションは以下の通りです。

- Ubuntu 13.04
- Ubuntu 13.10
- Ubuntu 14.04

インストールメディアおよびインストール手順については、以下のサイトを参考にしてください。

- Ubuntu Japanese Team  
<https://www.ubuntulinux.jp/>
- Ubuntu  
<http://www.ubuntu.com/>

## ROS

1. Ubuntu14.04 の場合は、下記の手順で ROS および必要なパッケージをインストールします。

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > \
/etc/apt/sources.list.d/ros-latest.list'
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install ros-indigo-desktop-full ros-indigo-velodyne-pointcloud \
ros-indigo-nmea-msgs
$ sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev
```

2. Ubuntu13.10 もしくは 13.04 の場合は、下記の手順で ROS および必要なパッケージをインストールします。

# sources.list の設定など必要

```
$ sudo apt-get install ros-hydro-desktop-full ros-hydro-velodyne-pointcloud \
ros-indigo-nmea-msgs
$ sudo apt-get install libnlopt-dev freeglut3-dev
```

3. ~/.bashrc などに以下を追加します。

```
[ -f /opt/ros/indigo/setup.bash ] && . /opt/ros/indigo/setup.bash
```

## OpenCV

OpenCV のサイト(<http://sourceforge.net/projects/opencvlibrary/>)からソースコードを入手し、

以下の手順でインストールを行います。

# 現在、2.4.8 が入手不可だが、他のバージョンでも OK か?

```
$ unzip opencv-2.4.8.zip
$ cd opencv-2.4.8
$ cmake .
$ make
$ sudo make install
```

## Qt

1. まず、Qt5 に必要なパッケージを、以下の手順でインストールします。

```
$ sudo apt-get build-dep qt5-default
$ sudo apt-get install build-essential perl python git
$ sudo apt-get install "libxcb.*" libx11-xcb-dev libglu1-mesa-dev \
libxrender-dev libxi-dev
$ sudo apt-get install flex bison gperf libicu-dev libxslt-dev ruby
```

```
$ sudo apt-get install libssl-dev libxcursor-dev libxcomposite-dev libxdamage-dev \
libxrandr-dev libfontconfig1-dev
$ sudo apt-get install libasound2-dev libgstreamer0.10-dev \
libgstreamer-plugins-base0.10-dev
```

2. 次に、Qt5 のソースコードを入手して、ビルドおよびインストールを行います。

```
$ git clone https://git.gitorious.org/qt/qt5.git qt5
$ cd qt5/
$ git checkout v5.2.1
$ perl init-repository --no-webkit
(webkit は大きいため、--no-webkit を指定しています)
$ ./configure -developer-build -opensource -nomake examples -nomake tests
(ライセンスを受諾する必要があります)
$ make -j
(ビルドには数時間かかります)
$ make install
$ sudo cp -r qtbase /usr/local/qtbase5
```

## CUDA

# <http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/> を参考に

1. 環境の確認

```
$ lspci | grep -i nvidia
(NVIDIA のボードの情報が出力されることを確認)
$ uname -m
(x86_64 であることを確認)
$ gcc --version
(インストールされていることを確認)
```

2. CUDA のインストール

<http://developer.nvidia.com/cuda-downloads> から CUDA をダウンロード

(以下、cuda-repo-ubuntu1404\_7.0-28\_amd64.deb と想定)

```
$ sudo dpkg -i cuda-repo-ubuntu1404_7.0-28_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

3. システムを再起動 (...は不要かもしれませんが)

```
$ lsmod | grep nouveau
(nouveau ドライバがロードされていないことを確認)
```

#### 4. 確認

```
$ cat /proc/driver/nvidia/version  
(カーネルモジュール、gcc のバージョンが表示される)  
$ cuda-install-samples-7.0.sh ~  
$ cd ~/NVIDIA_CUDA-7.0_Samples/1_Uutilities/deviceQuery/  
$ make  
$ ./deviceQuery
```

#### 5. CUDA を普段から使う場合は、以下の設定を .bashrc などを書く

```
export PATH="/usr/local/cuda:$PATH"  
export LD_LIBRARY_PATH="/usr/local/cuda/lib:$LD_LIBRARY_PATH"
```

## FlyCapture2

PointGray 社のカメラを使用する場合は、以下の手順で FlyCapture SDK をインストールします。

# 2014 年 10 月 28 日に試したときの手順

# /radisk2/work/usuda/autoware/doc/MultiCameraEclipse-log-20141028.txt

#### 1. PointGrey 社のサイト (<http://www.ptgrey.com/>) から、FlyCapture SDK をダウンロードします。(ユーザ登録が必要です。)

#### 2. 以下の手順で、事前にパッケージをインストールします。

```
$ sudo apt-get install libglademm-2.4-1c2a libgtkglextmm-x11-1.2-dev libserial-dev
```

#### 3. ダウンロードしたアーカイブを展開します。

```
$ tar xvfz flycapture2-2.6.3.4-amd64-pkg.tgz
```

#### 4. インストーラを起動します。

```
$ cd flycapture2-2.6.3.4-amd64/  
$ sudo sh install_flycapture.sh
```

This is a script to assist with installation of the FlyCapture2 SDK.

Would you like to continue and install all the FlyCapture2 SDK packages?

(y/n)\$ y ← 「y」 と答えます

...

Preparing to unpack updatorgui-2.6.3.4\_amd64.deb ...

Unpacking updatorgui (2.6.3.4) ...

updatorgui (2.6.3.4) を設定しています ...

Processing triggers for man-db (2.6.7.1-1ubuntu1) ...

Would you like to add a udev entry to allow access to IEEE-1394 and USB hardware?

If this is not ran then your cameras may be only accessible by running flycap as sudo.

(y/n)\$ y ← 「y」 と答えます

## Autoware

以下の手順で Autoware を入手し、ビルドおよびインストールを行います。

```
$ git clone https://github.com/CPFL/Autoware.git
$ cd Autoware/ros/src
$ catkin_init_workspace
$ cd ../
$ ./catkin_make_release
```

## AutowareRider

以下の URL から APK ファイルを入手し、インストールを行います。

- 本体
  - AutowareRider.apk  
<https://github.com/CPFL/Autoware/blob/master/ui/tablet/AutowareRider/AutowareRider.apk>
- 経路データ生成アプリケーション
  - AutowareRoute.apk  
<https://github.com/CPFL/Autoware/blob/master/ui/tablet/AutowareRoute/AutowareRoute.apk>
- CAN データ収集アプリケーション
  - CanDataSender.apk  
<https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CanDataSender/bin/CanDataSender.apk>
  - CanGather.apk  
<https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CanGather/apk/CanGather.apk>
  - CarLink\_CAN-BT\_LS.apk  
[https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink\\_CAN-BT\\_LS.apk](https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink_CAN-BT_LS.apk)
  - CarLink\_CANusbAccessory\_LS.apk  
[https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink\\_CANusbAccessory\\_LS.apk](https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink_CANusbAccessory_LS.apk)

CanGather は APK ファイル以外に、設定ファイルを用意する必要があります。

詳細は、以下の URL を参考にしてください。

<https://github.com/CPFL/Autoware/tree/master/vehicle/general/android#cangather-%E3%81%AE%E5%A0%B4%E5%90%88>

## canlib

kvaser のサイト(<http://www.kvaser.com/downloads/>) の "Kvaser LINUX Driver and SDK" よりソースコード linuxcan.tar.gz を入手し、以下の手順でインストールを行います。

```
$ tar xzf linuxcan.tar.gz
$ cd linuxcan
$ make
$ sudo make install
```

## SSH の公開鍵の作成

pos\_db はデータベースにアクセスする際に SSH の公開鍵認証を使用します。

そのため、pos\_db ノードを動かすユーザであらかじめ作成して公開鍵の情報を サーバに反映しておく必要があります。

### 1. 鍵の作成方法

- 以下のコマンドを実行して鍵を作成する
  - `$ ssh-keygen -t rsa`
- その際、パスフレーズは空(文字列を入力せずにエンターキーを押す)にして作成してください。
- DSA を使用する場合は `-t dsa` と指定してください。

### 2. 公開鍵をサーバーにコピーする

- 作成した公開鍵を以下のコマンドでサーバーにコピーします。
  - `$ ssh-copy-id -i ~/.ssh/id_rsa.pub posup@db3.ertl.jp`
- その際にパスワードを聞かれるので適宜入力してください。

## 使用手順

# デモなどで使われているものをいくつか説明  
# [https://github.com/CPFL/Autoware/wiki/5.-Moriyama-FOT-\(ja\)](https://github.com/CPFL/Autoware/wiki/5.-Moriyama-FOT-(ja))  
# 基本は Runtime Manager から制御  
# センサ, AutowareRider, AutowareTouch も使う例を入れる

## センサデータの取得

### 自動運転

## AutowareRider

### 概要

AutowareRider は、ROS PC で動作する Autoware をタブレット端末から操作するための、Knight Rider に似た UI を持った、Android アプリケーションです。

AutowareRoute は、MapFan SDK で実装された、経路データ生成のための Android アプリケーションです。

AutowareRider は、以下の機能を提供します。

- AutowareRoute で生成した経路データを ROS PC へ送信
- CAN データ収集アプリケーションを起動
- ボタン操作で ROS PC の Launch ファイルを起動
- ROS PC から受信した CAN データを UI へ反映

ここでは、これらの機能の使用手順を説明します。

## 起動方法

1. ROS PC で Runtime Manager を起動します。
2. Main タブ[Network Connection] - [Tablet UI]の Active ボタンを押下し、以下を起動します。
  - ui\_receiver
  - ui\_sender
3. Computing タブ[Planning] - [Path]の各アンカーから、以下を設定します。
  - lane\_navi
    - vector\_map\_directory  
高精度地図が格納されたディレクトリ
  - lane\_rule
    - vector\_map\_directory  
高精度地図が格納されたディレクトリ
    - ruled\_waypoint\_csv  
waypoint が保存されるファイル
    - Velocity  
速度（単位: km/h、初期値: 40、範囲: 0~200）
    - Difference around Signal  
信号の前後で加減速する速度（単位: km/h、初期値: 2、範囲: 0~20）
  - lane\_stop
    - Red Light  
赤信号時の速度へ切り替え
    - Green Light  
青信号時の速度へ切り替え
4. Computing タブ[Planning] - [Path]のチェックボックスを有効にし、以下を起動します。
  - lane\_navi
  - lane\_rule
  - lane\_stop
5. Android タブレットのアプリケーション一覧画面から AutowareRider を起動します。
6. [右上メニュー]→[設定]から、以下を設定します。
  - ROS PC
    - IP アドレス  
ROS PC IPv4 アドレス
    - 命令ポート番号  
ui\_receiver ポート番号 (初期値: 5666)



■ 情報ポート番号

ui\_sender ポート番号 (初期値: 5777)

7. [OK]を押下し、ROS PC へ接続を試みます。

- このとき設定はファイルに自動的に保存され、次の起動からは保存された設定で接続を試みます。

8. 画面中央のバーの色が、明るい赤で表示されている場合は接続に成功しています。

- バーの色と接続の状態

バーの色	接続の状態
暗い赤	ROS PC 未接続
明るい赤	ROS PC 接続
明るい青	自動運転 (mode_info: 1)
明るい黄	異常発生 (error_info: 1)

## 経路データ生成アプリケーションの使用方法

1. AutowareRider の NAVI ボタンを押下し、経路検索を起動します。
2. 地図を長押しして、以下を順番に実行します。
  - 出発地に設定
  - 目的地に設定
  - ルート探索実行
3. ルート探索の実行後に経路検索を終了することで、ROS PC へ経路データが転送されます。
  - このとき経路データはファイルに自動的に保存され、次回からはルート探索を省略して経路データを転送できます。
4. 転送後は、再び AutowareRider へ画面が戻ります。

## ROS PC への経路データ転送手順

上記の経路データ生成アプリケーションの使用方法 手順 3. を参照してください。

## CAN データ収集アプリケーションの使用方法

1. AutowareRider の[右上メニュー]→[設定]から、以下を設定します。  
これらの設定は AutowareRider から起動された、CanDataSender が使用します。

- データ収集
    - テーブル名  
データ転送先 テーブル名
  - SSH
    - ホスト名  
SSH 接続先 ホスト名
    - ポート番号  
SSH 接続先 ポート番号 (初期値: 22)
    - ユーザ名  
SSH でログインするユーザ名
    - パスワード  
SSH でログインするパスワード
  - ポートフォワーディング
    - ローカルポート番号  
ローカルマシンの転送元ポート番号 (初期値: 5558)
    - リモートホスト名  
リモートマシン ホスト名 (初期値: 127.0.0.1)
    - リモートポート番号  
リモートマシンの転送先ポート番号 (初期値: 5555)
2. [OK]を押下することで、設定がファイルに保存されます。
- ただし、SSH のパスワードはファイルに保存しません。AutowareRider を起動している間だけ、メモリにのみ保持しています。
3. [右上メニュー]→[データ収集]から、以下のいずれかを起動します。
- CanGather
  - CarLink (Bluetooth)
  - CarLink (USB)
4. アプリケーション起動後の使用方法は、それぞれを単独で起動した場合と同様です。
- 詳細は、以下の URL を参考にしてください。  
<https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/README.md>

## ROS PC への CAN データ転送手順

上記の CAN データ収集アプリケーションの使用法 手順 4. を参照してください。

## Launch ファイルの起動方法

1. AutowareRider の S1 ボタン、S2 ボタンは、それぞれが以下の Launch ファイルに対応しています。
  - check.launch
  - set.launch
2. ボタンを押下することで、ROS PC で Launch ファイルが起動します。
  - ボタンと Launch ファイルの状態

ボタン	Launch ファイルの状態
押下（文字色：黒）	起動（{\ndt, lf}_stat: false）
押下（文字色：赤）	起動（{\ndt, lf}_stat: true）

---

## 各機能の説明

### ROS

認知（物体検出，位置推定）

# ros/src/computing/perception/{detection,localization,...}

判断（レーン走行，交差点）

# ros/src/computing/planning/{mission, motion, path,...}

操作

# ros/src/computing/control

データ

# ros/src/data（ファイルや DB からデータを取得）

data のノード

1. points\_map\_loader  
path: ros/src/data/packages/map\_file/nodes/points\_map\_loader/  
publish\_msg: points\_map, pmap\_stat

subscribe\_msg: -  
parameter: -  
description: ポイントクラウド地図のファイルを読み込んで rviz に表示します

2. vector\_map\_loader

path: ros/src/data/packages/map\_file/nodes/vector\_map\_loader/  
publish\_msg: vector\_map, vmap\_stat, vector\_map\_info  
subscribe\_msg: -  
parameter: -  
argument: 高精度地図の CSV ファイル  
description: 高精度地図の CSV ファイルを読み込んで rviz に表示します

3. pos\_uploader

path: ros/src/data/packages/pos\_db/nodes/pos\_uploader/  
publish\_msg: -  
subscribe\_msg: current\_pose, car\_pose, pedestrian\_pose  
parameter: pos\_db/db\_host\_name  
pos\_db/db\_port  
pos\_db/sshpubkey  
pos\_db/sshprivatekey  
pos\_db/ssh\_port  
pos\_db/sshtunnelhost  
argument: データベースサーバのユーザ名,  
now(省略すると simulation time を使用)  
description: current\_pose, car\_pose, pedestrian\_pose のいずれかが publish された際  
に、それらの位置情報をデータベースサーバに登録を行う

4. pos\_downloader

path: ros/src/data/packages/pos\_db/nodes/pos\_downloader/  
publish\_msg: mo\_marker  
subscribe\_msg: -  
parameter: pos\_downloader/time  
pos\_downloader/delay  
pos\_downloader/life\_time  
pos\_downloader/posup\_dz  
pos\_downloader/pedestrian\_dz  
pos\_db/db\_host\_name  
pos\_db/db\_port  
pos\_db/sshpubkey  
pos\_db/sshprivatekey  
pos\_db/ssh\_port  
pos\_db/sshtunnelhost  
argument: データベースサーバのユーザ名,  
show\_my\_pose(省略すると自己車両を表示しない)

description: データベースサーバから取得した位置情報を rviz に表示します

## センサ

# ros/src/sensing/{calibration, drivers, fusion, sync, ...}

## 非 ROS モジュールとの通信

# ros/src/socket

### ui\_socket のノード

#### 1. ui\_receiver

path: ros/src/socket/packages/ui\_socket/nodes/ui\_receiver/

publish\_msg: gear\_cmd mode\_cmd route\_cmd

subscribe\_msg: -

parameter: ui\_receiver/port (default: 5666)

description: ROS 非対応の Android アプリケーションなどからのデータを ROS のメッセージに変換して publish するノードです。5666/TCP(パラメータで変更可能)で待ち受けます。

#### 2. ui\_sender

path: ros/src/socket/packages/ui\_socket/nodes/ui\_sender/

publish\_msg: -

subscribe\_msg: error\_info can\_info mode\_info ndt\_stat lf\_stat

parameter: ui\_sender/port (default: 5777)

description: ROS 非対応の Android アプリケーションなどへ、ROS のメッセージの情報を送信するノードです。5777/TCP(パラメータで変更可能)で待ち受けます。

### ui\_socket のメッセージ

#### 1. gear\_cmd

Header header

int32 gear

#### 2. mode\_cmd

Header header

int32 mode

#### 3. route\_cmd

Header header

Waypoint[] point

#### 4. Waypoint

float64 lat  
float64 lon

5. error\_info  
Header header  
int32 error

6. mode\_info  
Header header  
int32 mode

## ユーティリティ

# ros/src/util

## Runtime Manager

# ros/src/util/packages/runtime\_manager

### 概要

Runtime Manager は runtime\_manager パッケージに含まれる Python スクリプト (scripts/runtime\_manager\_dialog.py) を rosrun コマンドで起動し使用する。

```
$ rosrun runtime_manager runtime_manager_dialog.py
```

Runtime Manager を起動すると、画面にダイアログが表示される。

Runtime Manager のダイアログ操作により、

Autoware で使用する各種 ROS ノードの起動・終了処理や、  
起動した各種 ROS ノードへのパラメータ用のトピックの発行処理などを行なう事ができる。

Runtime Manager のダイアログの画面は、複数のタブ画面で構成される。

各種 ROS ノードを起動・終了するためのボタン類は、  
ノードの機能により、各タブ画面に分類・配置されている。

各タブ画面の表示は、画面上部のタブにより切替える。

Quick Start | Map | Sensing | Computing | Interface | Database | Simulation | Status

Map [ ] Ref

Sensing [ ] Ref

Localization [ ] Ref

Detection [ ] Ref

Mission Planning [ ] Ref

Motion Planning [ ] Ref

Android Tablet | Oculus Rift | Vehicle Gateway | Cloud Data

Auto Pilot | CPU0:12.5% CPU1:12.5% CPU2:12.7% CPU3:12.1% CPU4:6.4% CPU5:6.5% CPU6: MEM: 19GB/31GB(63%) | ROSBAG | Rviz

Runtime Manager 起動画面

## Quick Start タブ

### Map トグルボタン

Map テキストボックスで指定されている.launch スクリプトを起動・終了する。

### Map テキストボックス

Map トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

### Map Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Map テキストボックスに設定される。

### Sensing トグルボタン

Sensing テキストボックスで指定されている.launch スクリプトを起動・終了する。

### Sensing テキストボックス

Sensing トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

### Sensing Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Sensing テキストボックスに設定される。

### Localization トグルボタン

Localization テキストボックスで指定されている.launch スクリプトを起動・終了する。

### Localization テキストボックス

Localization トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

### Localization Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Localization テキストボックスに設定される。

### Detection トグルボタン

Detection テキストボックスで指定されている.launch スクリプトを起動・終了する。

### Detection テキストボックス

Detection トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

### Detection Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Detection テキストボックスに設定される。

### Mission Planning トグルボタン

Mission Planning テキストボックスで指定されている.launch スクリプトを起動・終了する。



#### Mission Planning テキストボックス

Mission Planning トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

#### Mission Planning Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Mission Planning テキストボックスに設定される。

#### Motion Planning トグルボタン

Motion Planning テキストボックスで指定されている.launch スクリプトを起動・終了する。

#### Motion Planning テキストボックス

Motion Planning トグルボタンから起動・終了させる.launch スクリプトのパスを指定する。  
(フルパスで指定する)

#### Motion Planning Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Motion Planning テキストボックスに設定される。

#### Android Tablet トグルボタン

ui\_socket/ui\_receiver, ui\_socket/ui\_sender ノードを  
起動・終了する。

#### Oculus Rift トグルボタン

〈未実装〉

#### Vehicle Gatewat トグルボタン

vehicle\_socket/vehicle\_receiver, vehicle\_socket/vehicle\_sender  
ノードを起動・終了する。

### Cloud Data トグルボタン

obj\_db/obj\_downloader ノードを起動・終了する。

### Auto Pilot トグルボタン

ボタンの状態に応じた mode\_cmd トピックを発行する。

### ROSBAG ボタン

ROSBAG Record ダイアログを表示する。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

### ROSBAG Record ダイアログ

上部テキストボックス

rosviz record コマンドを実行する際の、bag ファイルを指定する。

(フルパスで指定する)

### Ref ボタン

保存ファイル指定ダイアログが表示される。

指定したファイルは、上部テキストボックスに設定される。

### Start ボタン

上部テキストボックスに設定された bag ファイルを指定して、  
rosviz record コマンドを起動する。

### Stop ボタン

起動している rosviz record コマンドを終了する。

### All チェックボックス

チェックボックスが ON の場合、rosviz record コマンドを起動する際に、  
-a オプションが指定される。

### その他チェックボックス群

rosbag record コマンドを起動する際に、  
チェックボックスが ON のトピックを指定する。  
(ただし、All チェックボックスが OFF の場合のみ有効)

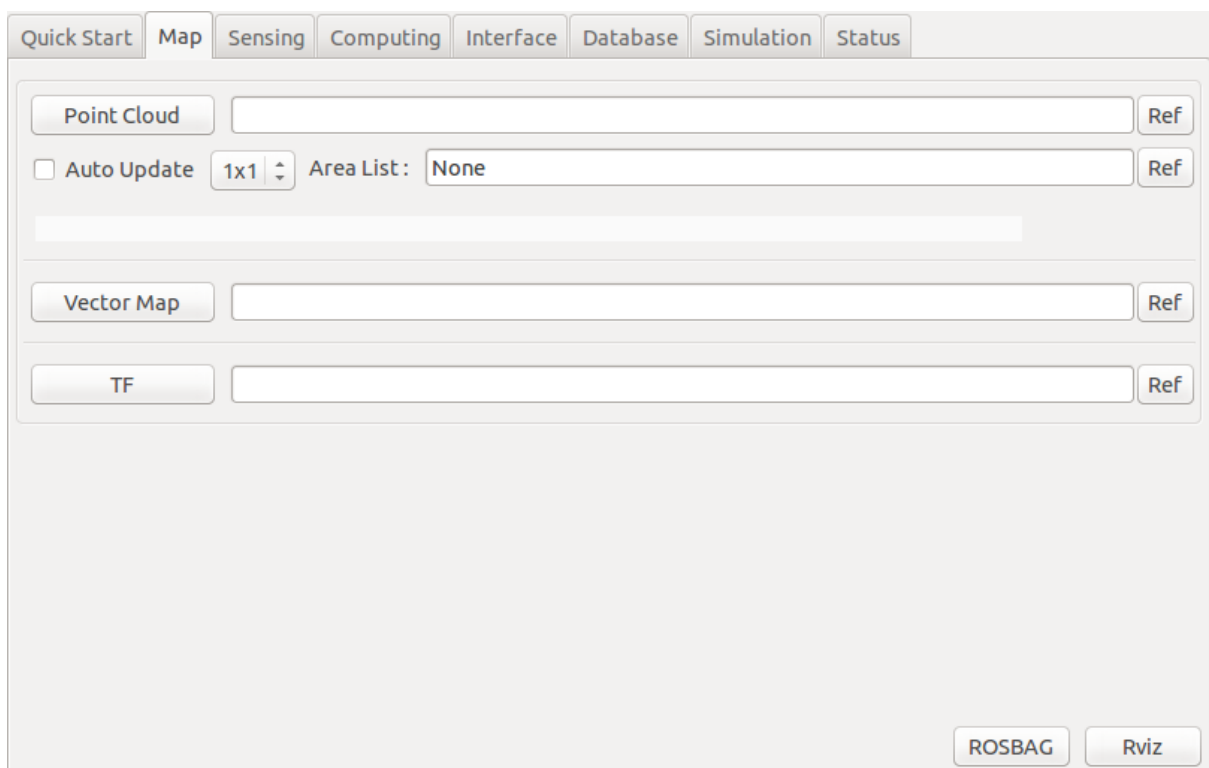
### Refresh ボタン

rostopic list コマンドを実行し、現在有効なトピックを調べ、  
その他のチェックボックス群を更新する。

### 最下行の情報表示

CPU(コア)の負荷状況とメモリ使用量を表示する。

## Map タブ



Map タブ

### Point Cloud トグルボタン

map\_file/points\_map\_loader ノードを起動・終了する。

### Point Cloud テキストボックス

Point Cloud トグルボタンで map\_file/points\_map\_loader を起動する際に引数で渡す、pcd ファイル群のパスを指定する。  
(フルパスを','で区切り指定する)

### Point Cloud Ref ボタン

ファイル選択ダイアログが表示される。  
複数のファイルが選択可能。(ただし同一ディレクトリに限る)  
選択したファイル群は、Point Cloud テキストボックスに設定される。

### Auto Update チェックボックス

Point Cloud トグルボタンで map\_file/points\_map\_loader を起動する際の、自動アップデートの有無を指定する

### Auto Update メニュー

Point Cloud トグルボタンで map\_file/points\_map\_loader を起動する際の、自動アップデート有効時の、シーン数を指定する。  
(Auto Update チェックボックスで ON が指定された場合のみ有効)

### Area List テキストボックス

Point Cloud トグルボタンで map\_file/points\_map\_loader を起動する際に引数で渡す、area list ファイルのパスを指定する。  
(フルパスで指定する)

### Area List Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、Area List テキストボックスに設定される。

### Vector Map トグルボタン

map\_file/vector\_map\_loader ノードを起動・終了する。

### Vector Map テキストボックス

Vector Map トグルボタンで map\_file/vector\_map\_loader を起動する際に引数で渡す、csv ファイル群のパスを指定する。  
(フルパスを','で区切り指定する)

### Vector Map Ref ボタン

ファイル選択ダイアログが表示される。  
複数のファイルが選択可能。(ただし同一ディレクトリに限る)  
選択したファイル群は、Vector Map テキストボックスに設定される。

### TF トグルボタン

TF テキストボックスに設定されている launch ファイルを起動・終了する。  
TF テキストボックスに launch ファイルが設定されていない場合は、  
次のパスの launch ファイルを起動・終了する。  
~/.autoware/data/tf/tf.launch

### TF テキストボックス

TF トグルボタンにより起動・終了させる launch ファイルのパスを指定する。  
(フルパスで指定する)

### TF Ref ボタン

ファイル選択ダイアログが表示される。  
選択したファイルは、TF テキストボックスに設定される。

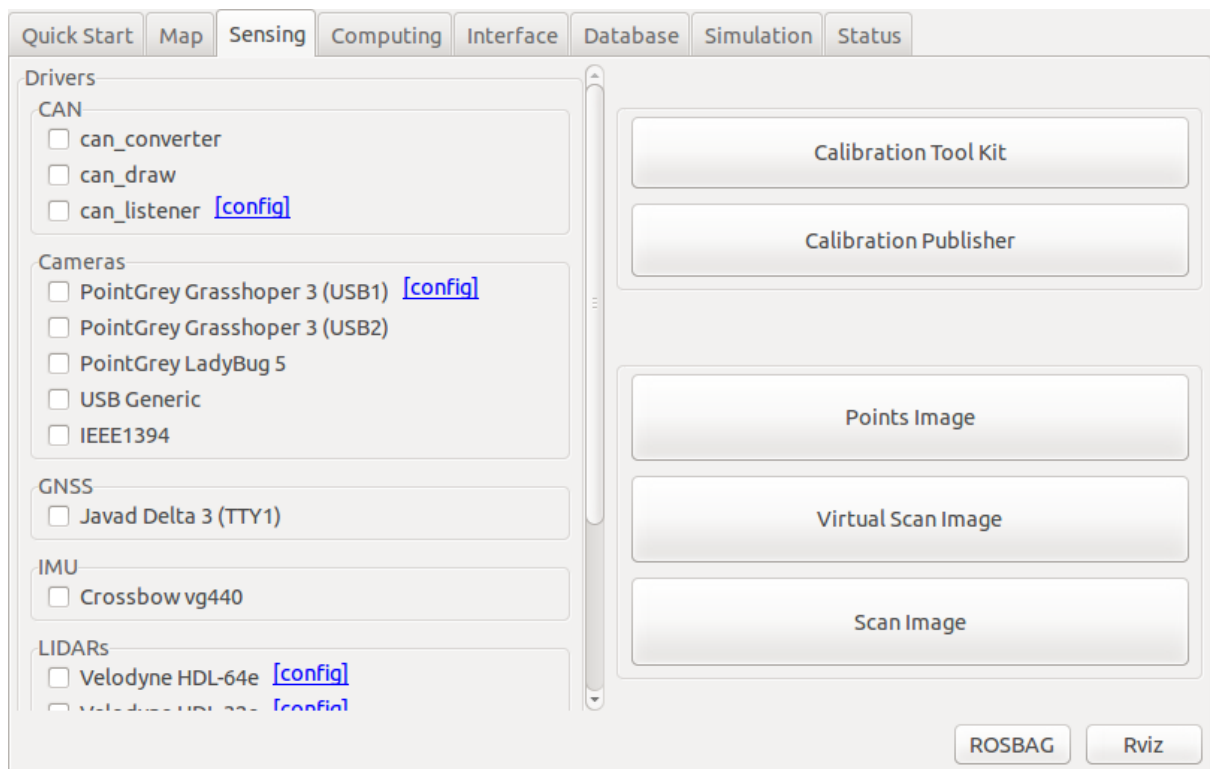
### ROSBAG ボタン

ROSBAG Record ダイアログを表示する。  
ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## **Sensing タブ**



Sensing タブ

#### Drivers/CAN 欄

can\_converter 項目

kvaser/can\_converter ノードを起動・終了する。

can\_draw 項目

kvaser/can\_draw ノードを起動・終了する。

can\_listener 項目

kvaser/can\_listener ノードを起動・終了する。

*config* リンク

can\_listener ダイアログを表示する。

ノード起動時に指定するチャンネルを設定する。

#### Drivers/Cameras 欄

PointGrey Grasshoper 3 (USB1)項目

pointgrey/grasshopper3.launch スクリプトを起動・終了する。

### *config* リンク

calibration\_path\_grasshopper3 ダイアログを表示する。

スクリプト起動時に指定する CalibrationFile の path を設定する。

PointGrey Grasshoper 3 (USB2)項目

〈未実装〉

PointGray LadyBug 5 項目

〈未実装〉

USB Generic 項目

uvc\_camera/uvc\_camera\_node ノードを起動・終了する。

IEEE1394 項目

〈未実装〉

### Drivers/GNSS 欄

Javad Delta 3 (TTY1)項目

javad/gnss.sh スクリプトを起動・終了する。

### Drivers/IMU 欄

Crossbow vg440 項目

〈未実装〉

### Drivers/LIDARs 欄

Velodyne HDL-64e 項目

velodyne/velodyne\_hdl64e.launch スクリプトを起動・終了する。

### *config* リンク

calibration\_path ダイアログを表示する。

スクリプト起動時に指定する calibration の path を設定する。

Velodyne HDL-32e 項目

velodyne/velodyne\_hdl32e.launch スクリプトを起動・終了する。

*config* リンク

calibration\_path ダイアログを表示する。

スクリプト起動時に指定する calibration\_path の値を設定する。

Hokuyo TOP-URG 項目

hokuyo/top\_urg スクリプト

Hokuyo 3D-URG 項目

hokuyo/hokuyo\_3d ノードを起動・終了する。

SICK LMS511 項目

〈未実装〉

IBEO 8L Single 項目

〈未実装〉

Drivers/OtherSensors 欄

〈項目なし〉

Calibration Tool Kti トグルボタン

camera\_lidar3d/camera\_lidar3d\_offline\_calib ノードを起動・終了する。

Calibration Publisher トグルボタン

calibration\_camera\_lidar/calibrtion\_publisher ノードを起動・終了する。

起動時に、calibration\_publiher ダイアログを表示するので、  
ノード起動時に指定する YAML ファイルのパスを設定する。  
(フルパスで指定する)

Points Image トグルボタン

points2image/points2image ノードを起動・終了する。

Scan Image トグルボタン

scan2image/scan2image ノードを起動・終了する。



### Virtual Scan Image トグルボタン

runtime\_manager/vscan.launch スクリプトを起動・終了する。

### ROSBAG ボタン

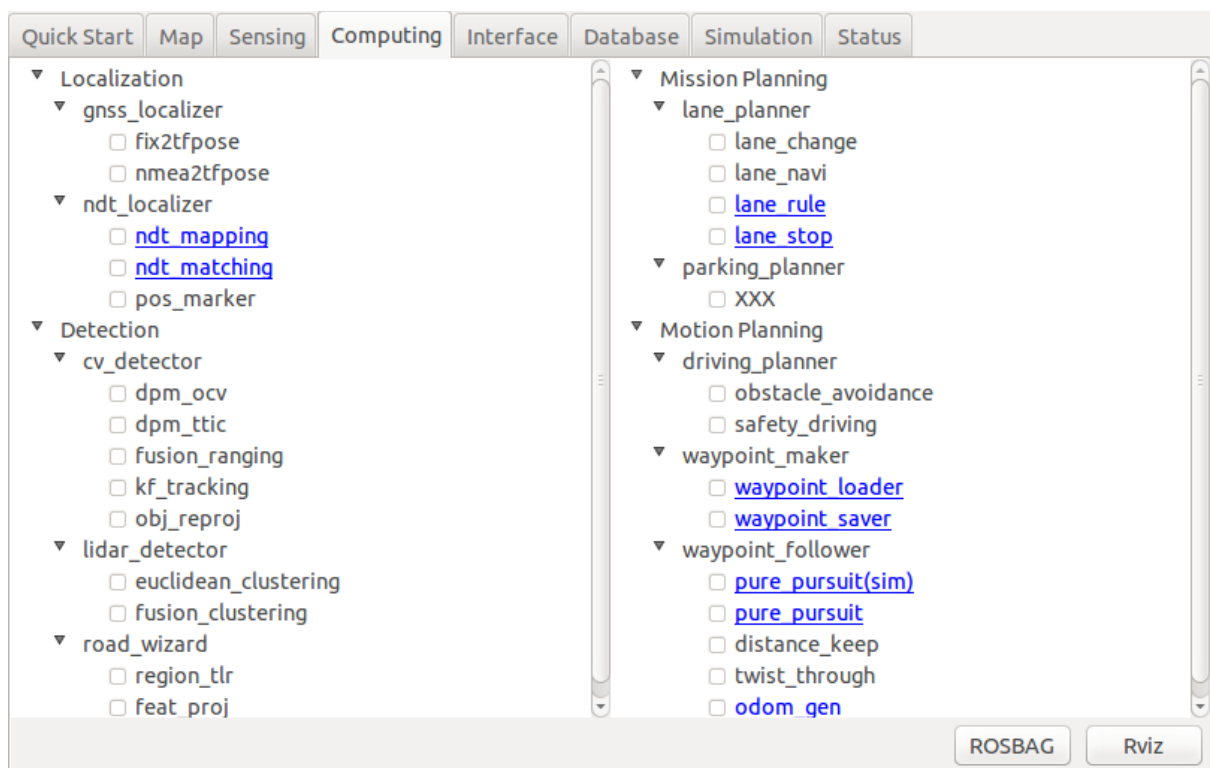
ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## Computing タブ



Computing タブ

### Localization/gnss\_localiser 欄

fix2tfpose 項目

gnss\_localizer/fix2tfpose ノードを起動・終了する。

nmea2tfpose 項目

gnss\_localizer/nmea2tfpose.launch スクリプトを起動・終了する。

Localization/ndt\_localiser 欄

ndt\_mapping 項目

ndt\_localizer/ndt\_mapping.launch スクリプトを起動・終了する。

リンク

ndt\_mapping ダイアログを表示する。

パラメータ変更後

/config/ndt\_mapping トピックを発行する。

ndt\_matching 項目

ndt\_localizer/ndt\_matching.launch スクリプトを起動・終了する。

リンク

ndt ダイアログを表示する。

パラメータ変更後

/config/ndt トピックを発行する。

Detection/cv\_detector 欄

dpm\_ocv 項目

＜未実装＞

dpm\_ttic 項目

＜未実装＞

fusion\_ranging 項目

＜未実装＞

kf\_tracking 項目

＜未実装＞

obj\_reproj 項目

＜未実装＞

## Detection/lidar\_detector 欄

euclidean\_clustering 項目

〈未実装〉

fusion\_clustering 項目

〈未実装〉

## Detection/road\_wizard 欄

region\_tlr 項目

〈未実装〉

feat\_proj 項目

〈未実装〉

## Mission Planning/lane\_planner 欄

lane\_change 項目

lane\_planner/lane\_chabge ノードを起動・終了する。

lane\_navi 項目

lane\_planner/lane\_navi ノードを起動・終了する。

lane\_rule 項目

lane\_planner/lane\_rule ノードを起動・終了する。

リンク

lane\_rule ダイアログを表示する。

パラメータ変更後

rosparam /lane\_rule/vector\_map\_directory,  
rosparam /lane\_rule/ruled\_waypoint\_csv を設定し、  
/config/lane\_rule トピックを発行する。

lane\_stop 項目

lane\_planner/lane\_stop ノードを起動・終了する。

リンク

lane\_stop ダイアログを表示する。

パラメータ変更後

/traffic\_light トピックを発行する。

#### Mission Planning/parking\_planner 欄

XXX 項目

＜未実装＞

#### Motion Planning/obstacle\_avoidance 欄

obstacle\_avoidance 項目

driving\_planner/obstacle\_avoidance ノードを起動・終了する。

safety\_driving 項目

driving\_planner/safety\_driving ノードを起動・終了する。

#### Motion Planning/waypoint\_maker 欄

waypoint\_loader 項目

waypoint\_maker/waypoint\_loader.launch スクリプトを起動・終了する。

##### リンク

waypoint\_loader ダイアログを表示する。

パラメータ変更後

/waypoint\_loader/vector\_map\_directory トピックを発行し、

rosparam /waypoint\_loader/ruled\_waypoint\_csv を設定し、

/config/waypoint\_loader トピックを発行する。

waypoint\_saver 項目

waypoint\_maker/waypoint\_saver.launch スクリプトを起動・終了する。

##### リンク

waypoint\_saver ダイアログを表示する。

スクリプト起動時に指定する save\_filename と Interval の値を設定する。

#### Motion Planning/waypoint\_follower 欄

pure\_pursuit(sim) 項目

lane\_follower/pure\_pursuit\_sim.launch スクリプトを起動・終了する。

## リンク

lane\_follower ダイアログを表示する。

パラメータ変更後

/config/lane\_follower トピックを発行する。

## pure\_pursuit 項目

lane\_follower/pure\_pursuit.launch スクリプトを起動・終了する。

## リンク

lane\_follower ダイアログを表示する。

パラメータ変更後

/config/lane\_follower トピックを発行する。

## distance\_keep 項目

waypoint\_follower/distance\_keep.launch スクリプトを起動・終了する。

## twist\_through 項目

waypoint\_follower/twist\_through ノードを起動・終了する。

## odom\_gen 項目

lane\_follower/odom\_gen.launch スクリプトを起動・終了する。

## リンク

odom\_gen ダイアログを表示する。

パラメータ変更後

/odom\_gen/use\_pose

/odom\_gen/initial\_pos\_x

/odom\_gen/initial\_pos\_y

/odom\_gen/initial\_pos\_z

/odom\_gen/initial\_pos\_roll

/odom\_gen/initial\_pos\_pitch

/odom\_gen/initial\_pos\_yaw

トピックを発行する。

## ROSBAG ボタン

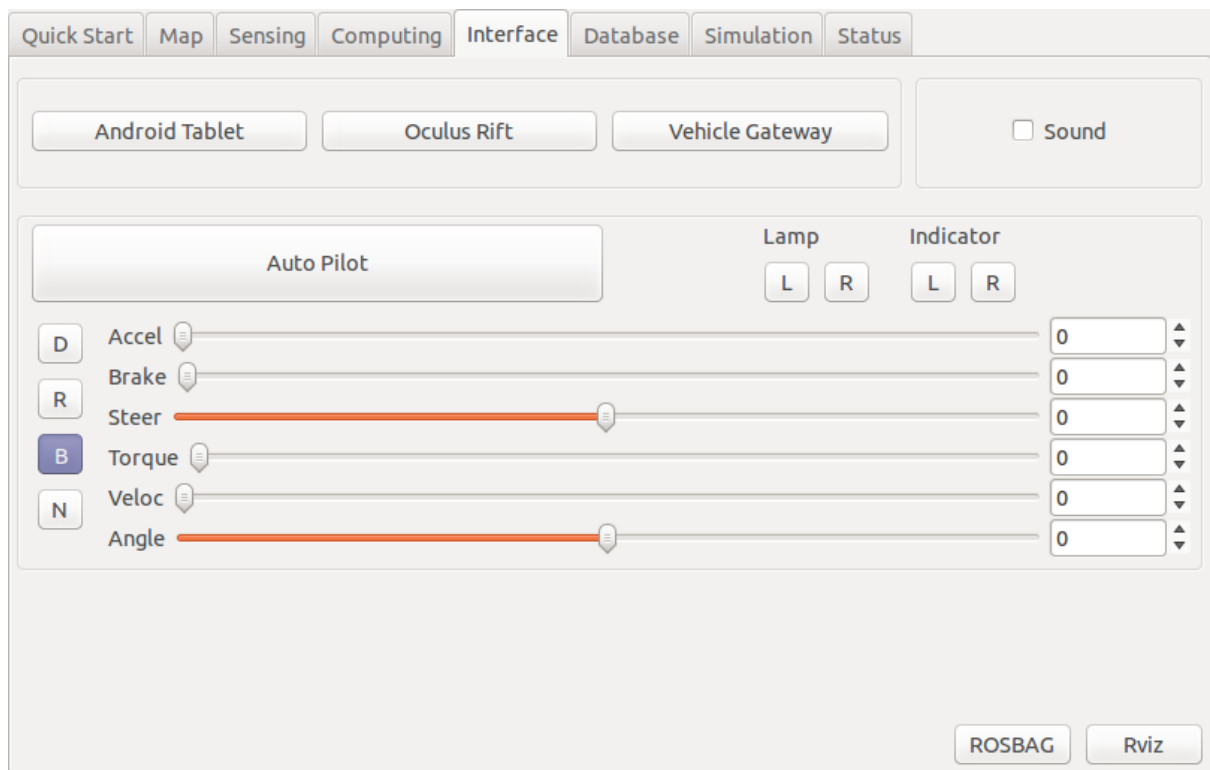
ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## Interface タブ



## Interface タブ

### Android Tablet トグルボタン

ui\_socket/ui\_receiver, ui\_socket/ui\_sender ノードを  
起動・終了する。

### Oculus Rift トグルボタン

〈未実装〉

### Vehicle Gatewat トグルボタン

vehicle\_socket/vehicle\_receiver, vehicle\_socket/vehicle\_sender  
ノードを起動・終了する。

### Sound チェックボックス

sound\_player/sound\_player.py スクリプトを起動・ 終了する。

### Auto Pilot トグルボタン

ボタンの状態に応じた mode\_cmd トピックを発行する。

### Lamp L, R トグルボタン

ボタンの状態に応じた lamp\_cmd トピックを発行する。

### Indicator L, R トグルボタン

ボタンの状態に応じた indicator\_cmd トピックを発行する。

### D,R,B,N ボタン

ON 操作したボタンに応じた gear\_cmd トピックを発行する。

### Accel スライダー

accel\_cmd トピックを発行する。

### Brake スライダー

brake\_cmd トピックを発行する。

### Steer スライダー

steer\_cmd トピックを発行する。

### Torque スライダー

〈未実装〉

### Veloc スライダー

twist\_cmd トピックを発行する。

(スライダーの値はメッセージの twist.linear.x フィールドに反映)

### Angle スライダー

twist\_cmd トピックを発行する。

(スライダーの値はメッセージの twist.angular.z フィールドに反映)

### ROSBAG ボタン

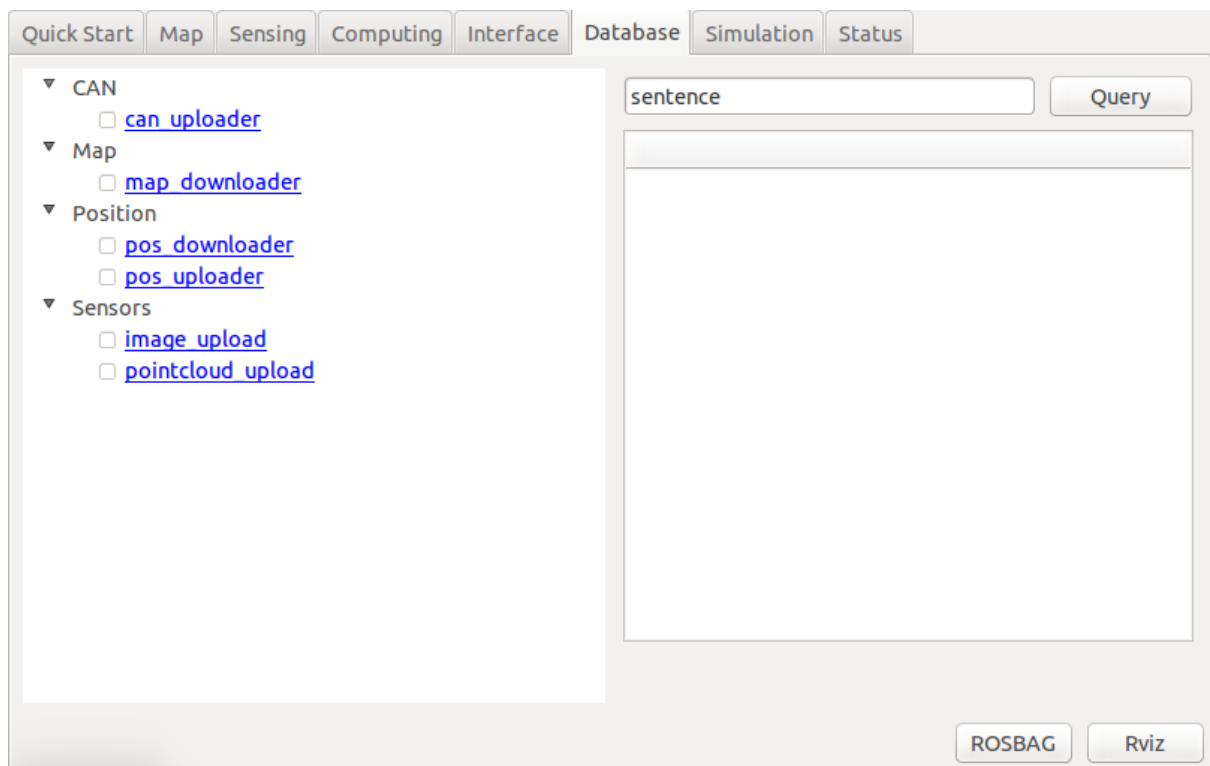
ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## Database タブ



Database タブ

### CAN 欄

can\_uploader 項目

obj\_db/can\_uploader ノードを起動・終了する。



リンク

other ダイアログを表示する。

#### Map 欄

map\_downloader 項目

〈未実装〉

リンク

map\_file ダイアログを表示する。

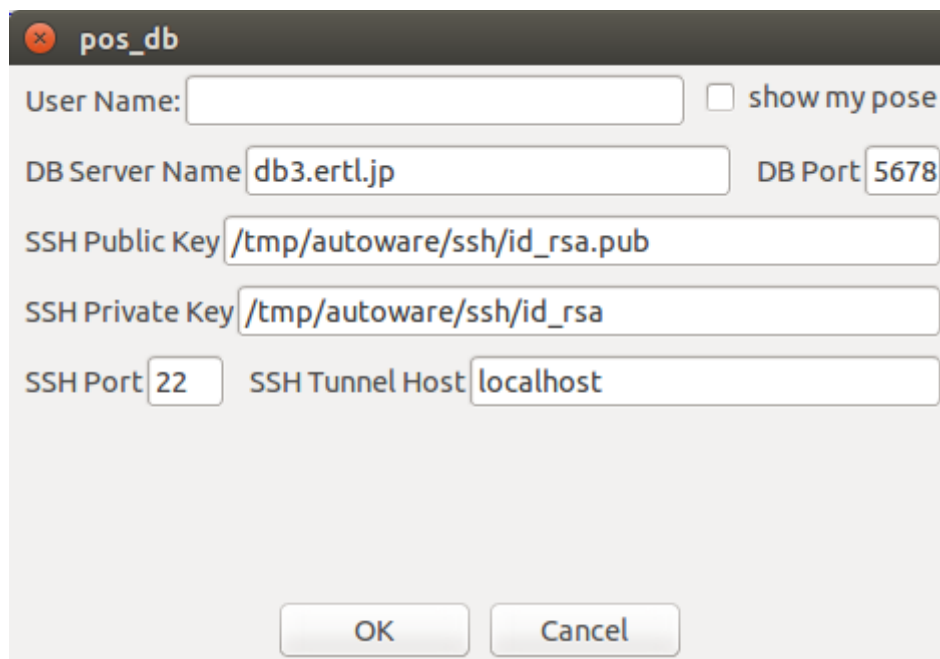
#### Position 欄

pos\_downloader 項目

pos\_db/pos\_downloader ノードを起動・終了する。

リンク

pos\_db ダイアログを表示する。



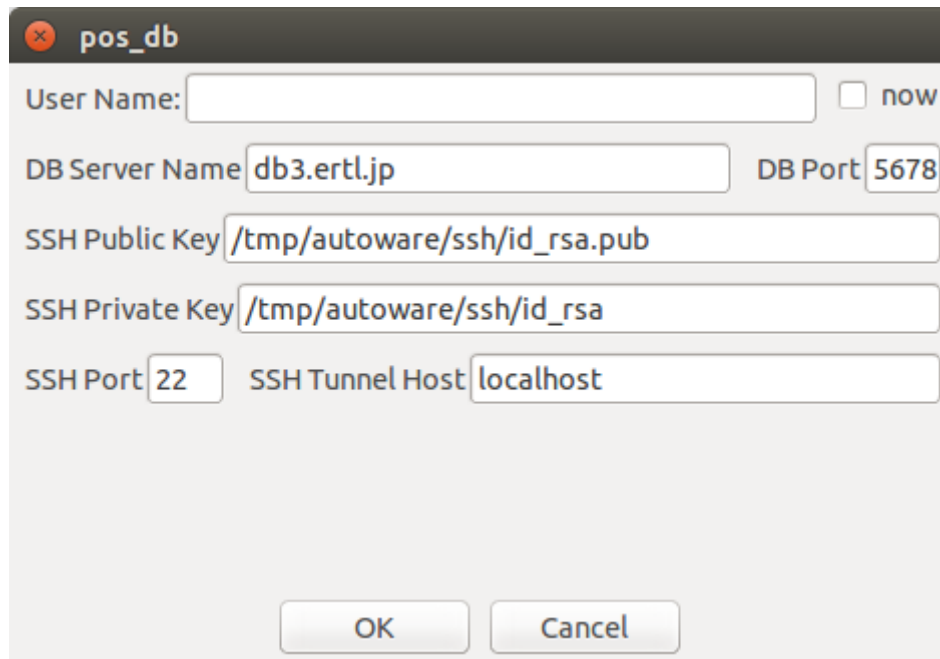
pos\_db ダイアログ(pos\_downloader)

pos\_uploader 項目

pos\_db/pos\_uploader ノードを起動・終了する。

リンク

pos\_db ダイアログを表示する。



pos\_db

User Name:  ☐ now

DB Server Name  DB Port

SSH Public Key

SSH Private Key

SSH Port  SSH Tunnel Host

OK Cancel

pos\_db ダイアログ(pos\_uploader)

#### Sensors 欄

image\_upload 項目

〈未実装〉

リンク

other ダイアログを表示する。

pointcloud\_upload 項目

〈未実装〉

リンク

other ダイアログを表示する。

#### Query テキストボックス

〈未実装〉

#### Query ボタン

〈未実装〉

#### ROSBAG ボタン

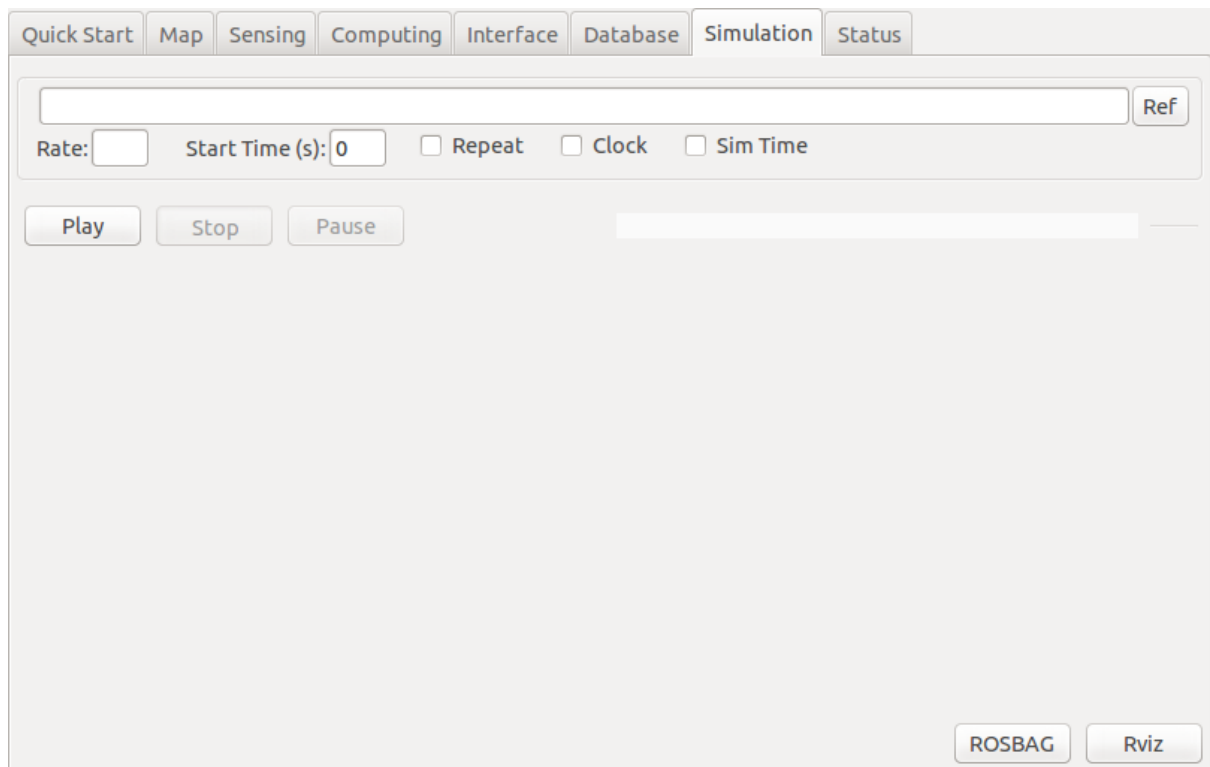
ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## Simulation タブ



Simulation タブ

### ROSBAG テキストボックス

Play ボタンで rosbag play コマンドを実行する際の、bag ファイルを指定する。  
(フルパスで指定する)

### ROSBAG Ref ボタン

ファイル選択ダイアログが表示される。

選択したファイルは、ROSBAG テキストボックスに設定される。

### Rate テキストボックス

rosbag play コマンドを起動する際の -r オプションで指定する数値を指定する。

未設定の場合は -r オプションを指定しない。

### Start Time(s) テキストボックス

rosbag play コマンドを起動する際の --start オプションで指定する開始位置の秒数を指定する。

未設定の場合は --start オプションを指定しない。

### Repeat チェックボックス

チェックボックスが ON の場合、rosbag play コマンドを起動する際に、  
--loop オプションが指定される。

### Clock チェックボックス

チェックボックスが ON の場合、rosbag play コマンドを起動する際に、  
--clock オプションが指定される。

### Sim Time チェックボックス

rosparam /use\_sim\_time の設定値 (true,false) を表示する。

チェックボックスを操作すると、値を rosparam /usr\_sim\_time に設定する。

### Play ボタン

ROSBAG テキストボックスに設定された bag ファイルを指定して、  
rosbag play コマンドを起動する。

### Stop ボタン

起動している rosbag play コマンドを終了する。

### Pause ボタン

起動している rosbag play コマンドを一時停止する。

## ROSBAG ボタン

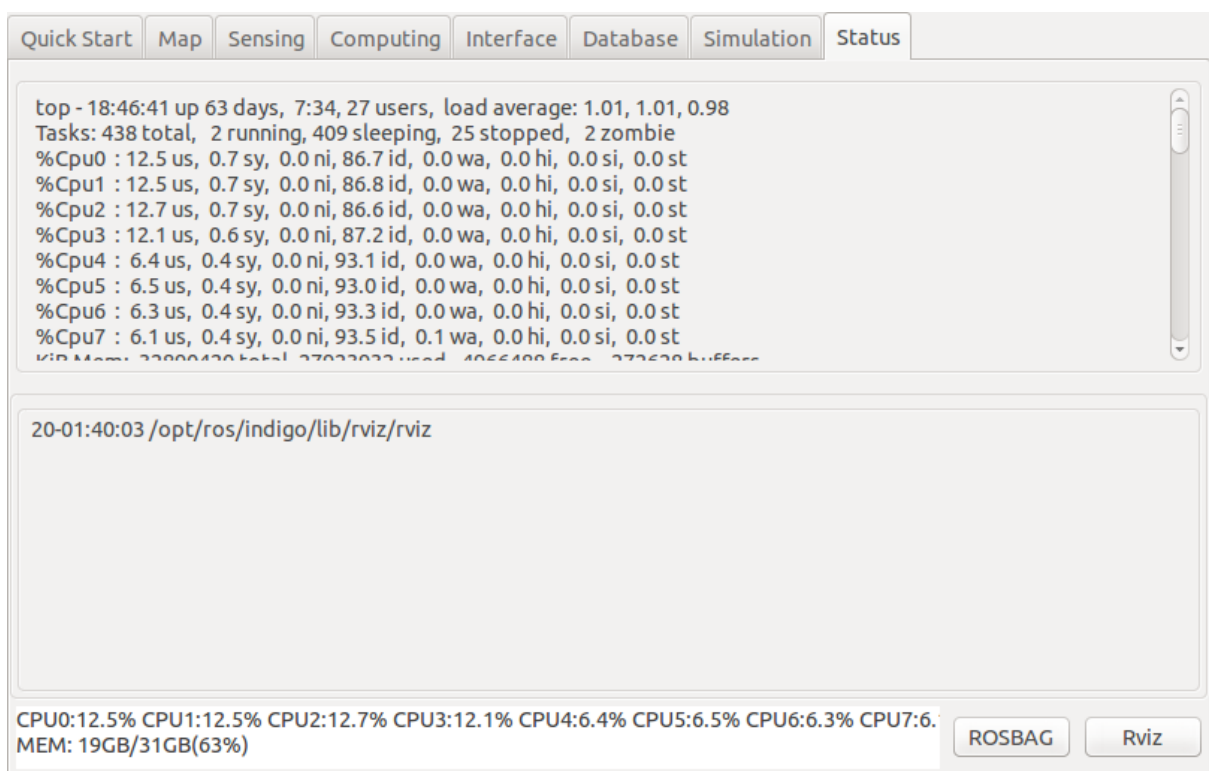
ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

## Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## Status タブ



## Status タブ

### 上段の表示

内部で実行している top コマンドの実行結果を表示する。

### 下段の表示

登録ノードの起動してからの経過時間を表示する。

#### 最下行の情報表示

CPU(コア)の負荷状況とメモリ使用量を表示する。最下行の情報表示

CPU(コア)の負荷状況とメモリ使用量を表示する。最下行の情報表示

CPU(コア)の負荷状況とメモリ使用量を表示する。（仕様の変更予定有り）

### 最下行の情報表示

CPU(コア)の負荷状況とメモリ使用量を表示する。

### ROSBAG ボタン

ROSBAG Record ダイアログを表示する。

ROSBAG Record ダイアログの詳細は Quick Start タブを参照。

### Rviz トグルボタン

rviz/rviz ノードを起動・終了する。

## ユーザインタフェース

### 概要

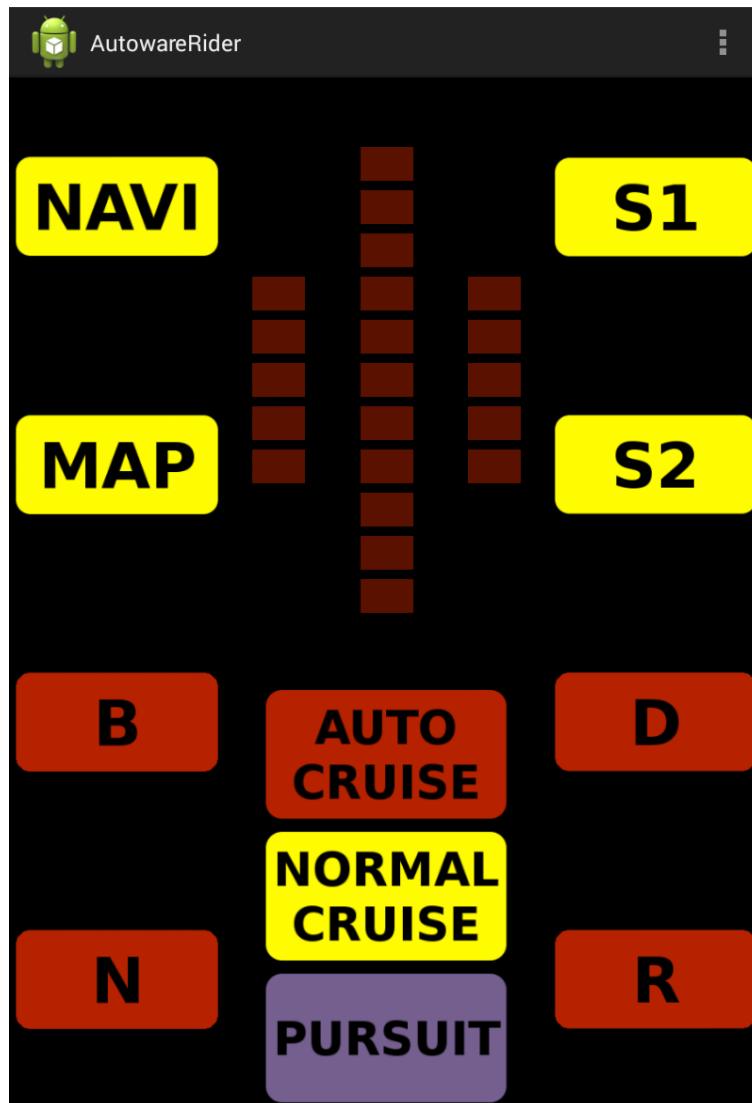
AutowareRider は、ROS PC で動作する Autoware をタブレット端末から操作するための、Knight Rider に似た UI を持った、Android アプリケーションです。

AutowareRoute は、MapFan SDK で実装された、経路データ生成のための Android アプリケーションです。

ここでは、これらの UI の機能を説明します。

### AutowareRider

以下が起動時の画面です。



図の各ボタンの機能は以下です。

- NAVI
  - AutowareRoute.apk の起動
- MAP
  - 未実装
- S1
  - check.launch を ROS PC で起動
- S2
  - set.launch を ROS PC で起動
- B
  - ギア情報 B を ROS PC へ送信
- N
  - ギア情報 N を ROS PC へ送信
- D
  - ギア情報 D を ROS PC へ送信

- R
  - ギア情報 R を ROS PC へ送信
- AUTO CRUISE
  - 未実装
- NORMAL CRUISE
  - 未実装
- PURSUIT
  - 未実装（現状はアプリケーションの終了）

[右上メニュー]から以下が選択できます。

- [設定]
- [データ収集]

以下が[設定]の画面です。



設定

ROS PC

IPアドレス: 192.168.0.10

命令受信ポート番号: 5666

情報送信ポート番号: 5777

データ収集

テーブル名: candata

SSH

ホスト名: candb.jp

ポート番号: 22

ユーザ名: autoware

パスワード: .....

ポートフォワーディング

ローカルポート番号: 5558

リモートホスト名: 127.0.0.1

リモートポート番号: 5555

キャンセル OK

図の各項目の説明は以下です。



- ROS PC
  - IP アドレス  
ROS PC IPv4 アドレス
  - 命令ポート番号  
ui\_receiver ポート番号 (初期値: 5666)
  - 情報ポート番号  
ui\_sender ポート番号 (初期値: 5777)
- データ収集
  - テーブル名  
データ転送先 テーブル名
- SSH
  - ホスト名  
SSH 接続先 ホスト名
  - ポート番号  
SSH 接続先 ポート番号 (初期値: 22)
  - ユーザ名  
SSH でログインするユーザ名
  - パスワード  
SSH でログインするパスワード
- ポートフォワーディング
  - ローカルポート番号  
ローカルマシンの転送元ポート番号 (初期値: 5558)
  - リモートホスト名  
リモートマシン ホスト名 (初期値: 127.0.0.1)
  - リモートポート番号  
リモートマシンの転送先ポート番号 (初期値: 5555)

以下が[データ収集]の画面です。



図の各ボタンの機能は以下です。

- CanGather
  - CanGather.apk の起動
- CarLink (Bluetooth)
  - CarLink\_CAN-BT\_LS.apk の起動
- CarLink (USB)
  - CarLink\_CANusbAccessory\_LS.apk の起動

## AutowareRoute

以下が起動時の画面です。



地図を長押しすることで、以下のダイアログが表示されます。



図の各ボタンの機能は以下です。

- 出発地に設定
  - 長押しした地点を経路データの出発地として設定
- 立寄地に設定
  - 長押しした地点を経路データの立寄地として設定
- 目的地に設定
  - 長押しした地点を経路データの目的地として設定
- ルート消去
  - ルート探索実行によって生成された経路データの消去
- ルート探索実行
  - 出発地、立寄地、目的地に応じた経路データの生成

車の制御

一般

# vehicle/general

**ZMP**

# vehicle/zmp