Version
0.9

名古屋大学

Autoware デベロッパーズマニュアル

Autoware ver.2015.12.12

目次

チャプター

1

1. このドキュメントについて

ドキュメントの位置付けについて説明します。

名古屋大学が無償で提供する Autoware に関するドキュメントは、以下の２つです。

「Autoware **ユーザーズ マニュアル**」

「Autoware **デベロッパーズ マニュアル**」

上記ドキュメントには日本語版です。今後、それぞれの英語版が作成される予定です。

「Autoware User's Manual」

「Autoware Developer's Manual」

チャプター

2

2. ROS と Autoware

Autoware の操作をする前に、理解を深めるために Autoware のベースである ROS の概要と Autoware の概要について説明します。

ロボット用のミドルウエア ROS

近年、ロボットの多様な可能性が着目されており、専門家以外からのロボット開発参入によって、その技術だけでなく、様々な分野への発展へとつなげることができると期待されます。しかし、ロボット開発はその機能の高度化と複雑化により、より難易度が増しています。パソコンやスマートフォンと異なり、それぞれのハードや OS、プログラム言語が異なることなども、ロボットの専門家はもとより、ロボット開発者の新規参入を妨げる大きな理由となっていました。

このことから、共通プラットフォーム化への期待が高まり、様々なプラットフォームが作成されています。プラットフォームが共通となることで、他者が作成したソフトウェアを部品のように組み合わせ、再利用することで開発期間を短縮し、開発者が各自の得意とする分野研究をより探究できるようになることが期待できます。

ROS（Robot Operating System）は米 Willow Garage が開発し、OSFR（Open Source Robotics Foundation）が維持・管理している、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です[1]。オープンソース化された ROS は欧米を中心に、日本国内でも普及しはじめています。

ROS は名前に「OS」と付いてはいますが、Windows や Linux のような OS とは異なり、Unix ベースの OS 上で動作するミドルウェアです。

---

1 https://en.wikipedia.org/wiki/Willow_Garage

ROS の特徴

1. ライブラリとツールの提供

ROS ではロボット用のソフトウェアを開発する為のライブラリとツールが提供されます。以下は主なライブラリとツールです。

- ✓ 独自のビルドシステム（Catkin）
- ✓ 画像処理ライブラリ（OpenCV）
- ✓ データロギングツール（rosbag）
- ✓ データとソフトウェア状態の視覚化ツール（rviz）
- ✓ 座標変換ライブラリ（tf）
- ✓ Qt ベースの GUI ソフト開発ツール（tqt）

2. プロセス間通信

ROSはモジュール間接続および連携のフレームワークに「Topic」を用いたPublish /Subscribe型のメッセージパッシングを用いています。メッセージパッシングとは 1 つ以上の受信者に対して送信者がデータを配信するプロセス間通信の方式です。この方式により分散型システムへの拡張ができます。

ROS では「Node」と呼ばれるプロセスを複数立ち上げ、各 Node が独立して実行されます。Node 間では Topic に「Message」と呼ぶデータを書き込み（Publish）し、その Topic を購読している Node が書き込まれた Message を読み込み（Subscribe）ます。

3. 構成要素

・バッグファイル（rosbag）

ROS では全ての Topic 上のメッセージがタイムスタンプと共に「rosbag」と呼ぶ.bag ファイルに記録でき、rviz で記録時と同じタイミングで再生することができます。多様なセンサを用いるロボット開発では、各種センサの動作情報の時間を同期させて総合的に解析するのは難しいため、ROS ではこれによって効率的に不具合などの解析ができます。また、記録した処理を繰り返し入力することもできるので、カメラやセンサが無くてもデバッグすることができます。

・Launch ファイル

複数の Node を一度に起動するには、Launch ファイルを用意してそれを実行することができます。Launch ファイルには起動したい Node の情報など起動に必要な情報を XML 形式で記述します。

Autoware

Autoware は、ROS 上で動くオープンソースソフトウェアです。自動運転技術の研究開発用に GitHub 上で公開しています。

自動運転システムの多くは「認知」、「判断」、「操作」で構成されています。

Autoware では 3 次元地図生成、自己位置推定、物体検出、走行制御といった、自動運転に必要な機能を提供しています。

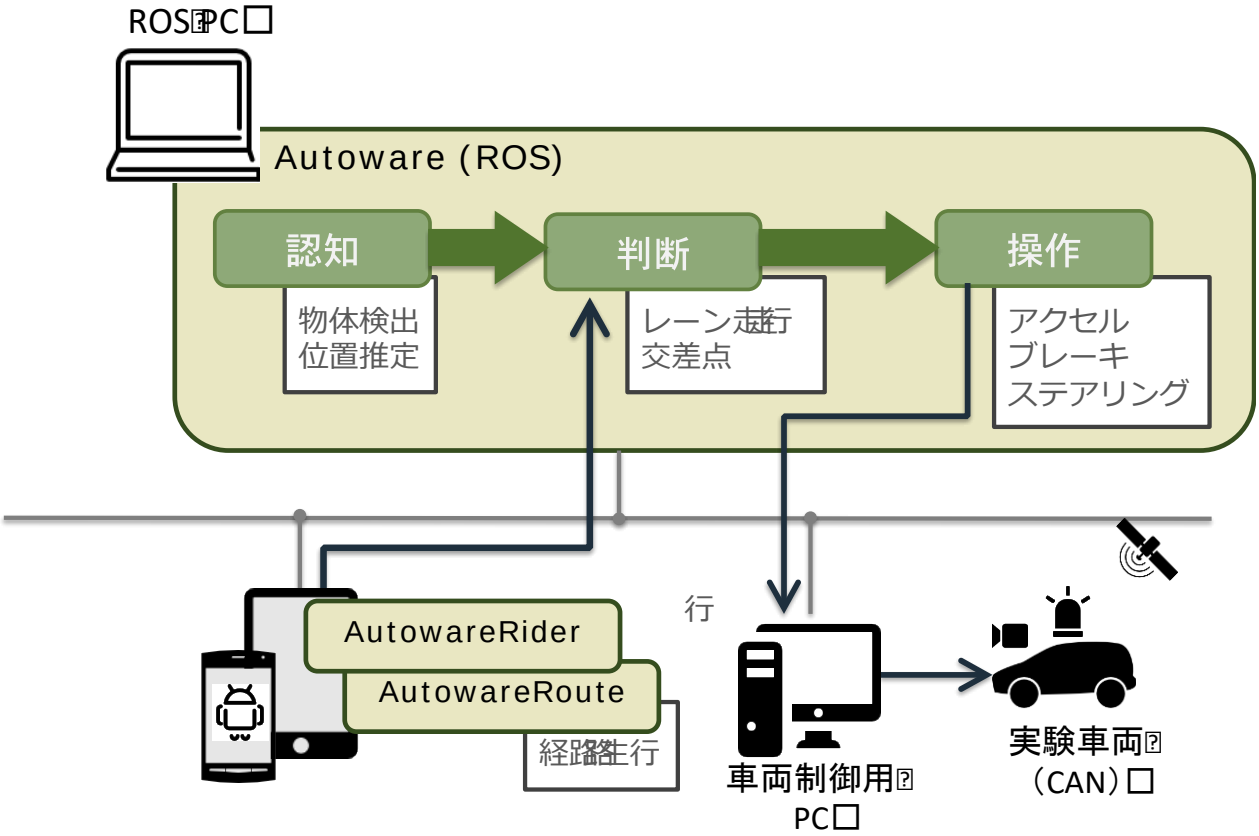


図 1 Autoware 全体構成

Autoware では、LIDAR や車載カメラを組み合わせて自車位置の推定をし、LIDAR と GNSS を用いて歩行者、周囲の車両、信号機などの周囲の物体検出を行ないます。

レーンや交差点での走行・停止といった「判断」には組込みメニーコア CPU を用いて実現しています。
実際の車両の「操作」にあたる車両制御については従来の自動車に搭載されている機能を用いて実現しています。

3 次元地図の作成と共有

Autoware の最大の特徴は 3 次元地図を基に自動運転を行う点にあります。3 次元地図とは、一般的なカーナビゲーションなどで使われている 2 次元の地図とは違い、道路周辺に設置されている立体物などを含めてさまざまな情報が取り込まれた地図です。今後、自動運転技術を実用化するには、3 次元地図の普及が重要な役割を果たします。
Autoware では、自車位置の推定は Autoware を動作させる PC にあらかじめ取り込んでおいた 3 次元地図と、車両上部に設置した LIDAR が取得する車両周辺の情報を照らし合わせることで、自車の位置を把握する仕組みになっています。その精度は約 10cm と、GPS よりもはるかに高精度です。
2015 年 9 月時点では、3 次元地図があるのはごく一部の地域だけです。もし Autoware を搭載した自動運転車が 3 次元地図が用意されていない区域に入った場合には、LIDAR が取得する情報から、リアルタイムで新たに 3 次元地図を生成することができます。

このようにして生成された 3 次元地図は、Autoware を使用して開発する開発者自身の自動運転技術開発のために活用するだけでなく、Autoware の機能を用いて名古屋大学のサーバにアップロードして共有することができます。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。この共有の仕組みによって 3 次元地図の作製専用車では入れないような細かな路地など、市街地のすみずみまで 3 次元地図を用意できるようになります。
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。

自己位置推定（NDT：Normal Distributions Transform）

自車位置は、Point Cloud と 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。

物体検出

カメラ画像を入力として、DPM（Deformable Part Models）アルゴリズムによる画像認識を行うことで、車や歩行者、信号機を検出できます。カルマンフィルタを利用したトラッキングも可能で、検出精度を高めています。また、3 次元 LIDAR データをフュージョン（融合）することで検出した物体の距離も取得できます。

信号機も検出できます。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、その 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、そこから画像処理によって色判別することで信号機検出を実現しています。

経路生成

目的地までのルートは AutowareRoute（MapFan を使用した経路データ生成アプリケーション）で生成し、自動運転システムが、そのルート上で使用する車線を決定します。経路データ生成アプリケーションは、スマートフォンのカーナビアプリケーションとして提供されています。車線内では運動学的に導かれた軌跡を生成します。

自律走行

生成した経路には適切な速度情報も含まれており、その速度を目安に自律走行します。また、経路には 1m 間隔で設定された目印の「way point」が設定してあり、それ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、自律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。もっとも安全な経路を選択して走行します。

ユーザインタフェース

Autoware のデベロッパ用ユーザインタフェースである「Runtime Manager」を利用する

ことで、位置推定や物体検出、経路追従などの処理を簡単に操作できます。

rviz を用いて 3 次元地図上での自己位置推定、物体検出、経路生成、経路追従を統合し、可視化できます。

Autoware のユーザ用タブレットユーザインタフェースである「AutowareRider」を利用すると、タブレットでナビや経路生成、自動運転モードへの移行などの処理を簡単に操作できます。

自動運転システムが使っている 3 次元地図情報を可視化し、車載ディスプレイや Oculus デバイスに投影することもできます。

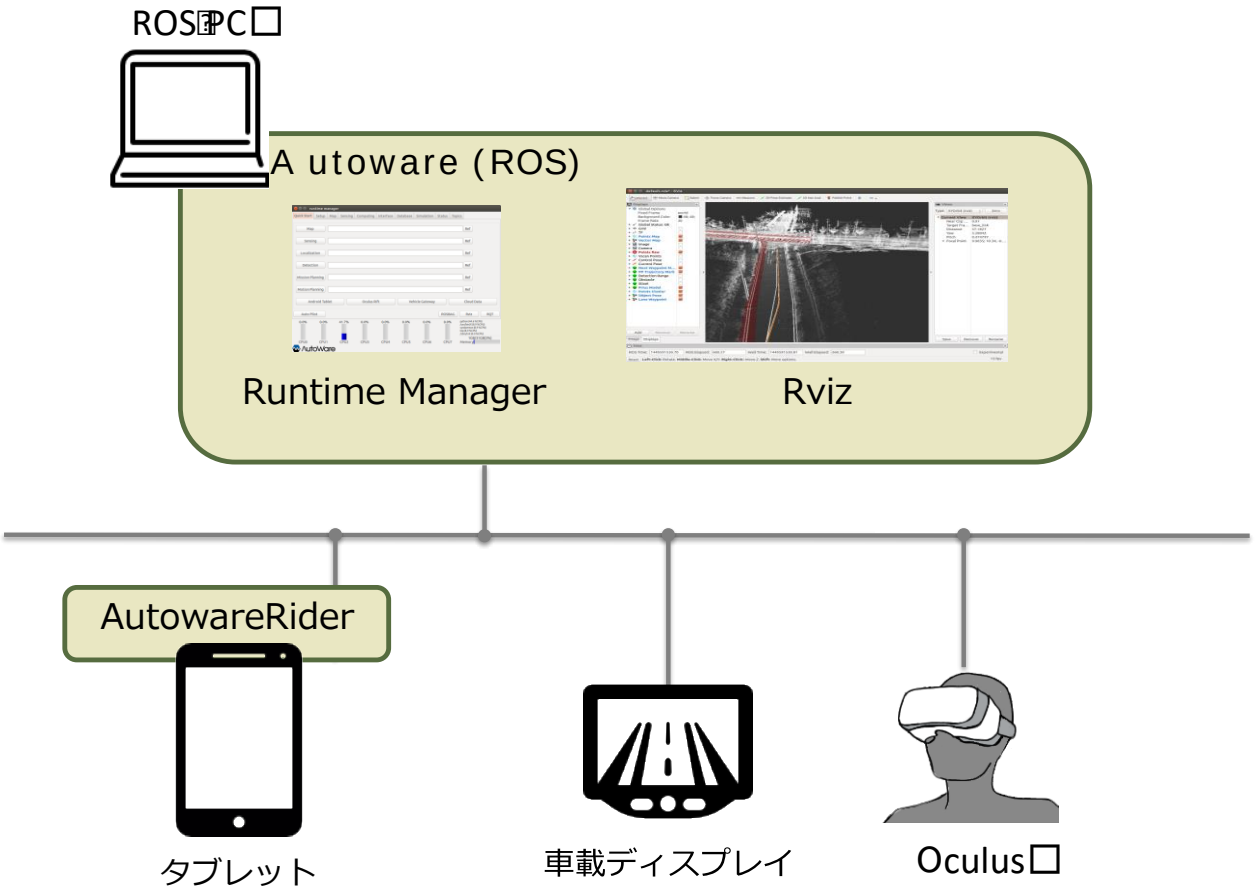


図 2 Autoware のユーザインタフェース

ROS と Autoware

ROS と Autoware の構成は以下のようになっています。

ROS は Unix ベースのプラットフォームでのみ、動作します。

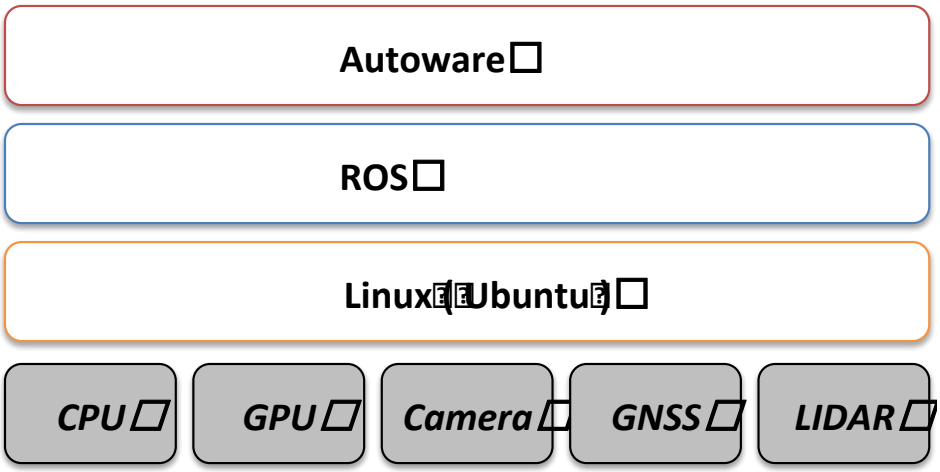


図 3 ROS と Autoware のモデル

Autoware の構造

認知・判断

ros/src/computing/perception/detection

road_wizard, cv_tracker, lidar_tracker

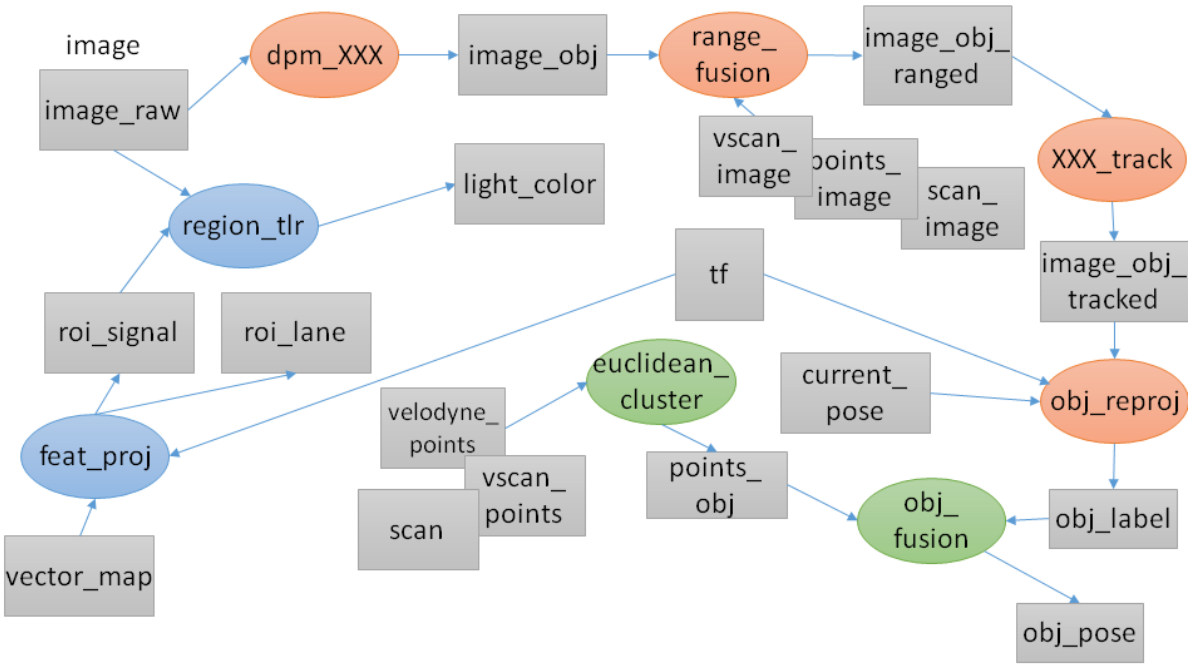


図 4 認知・判断

判断・操作・位置推定

ros/src/computing/perception/localization

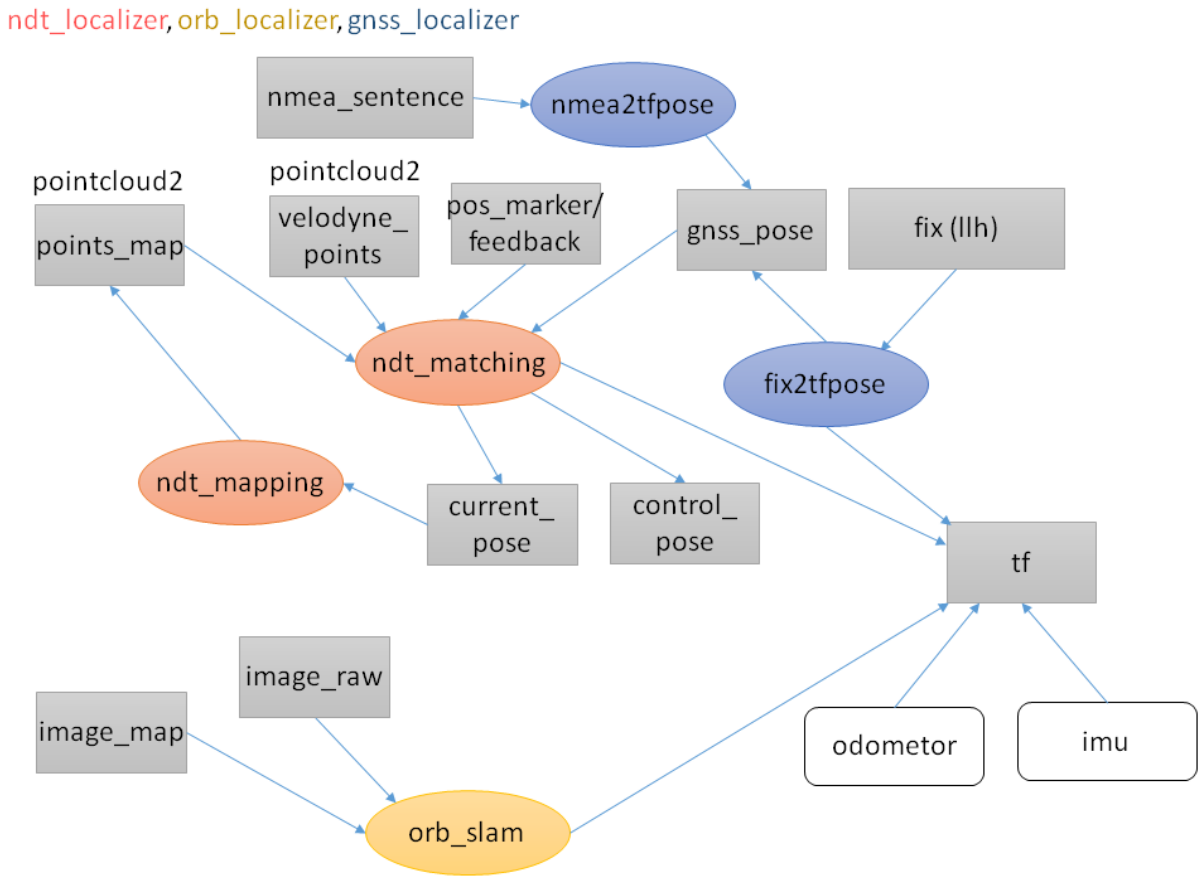


図 5 判断･操舵･位置推定

経路探索

ros/src/computing/planning

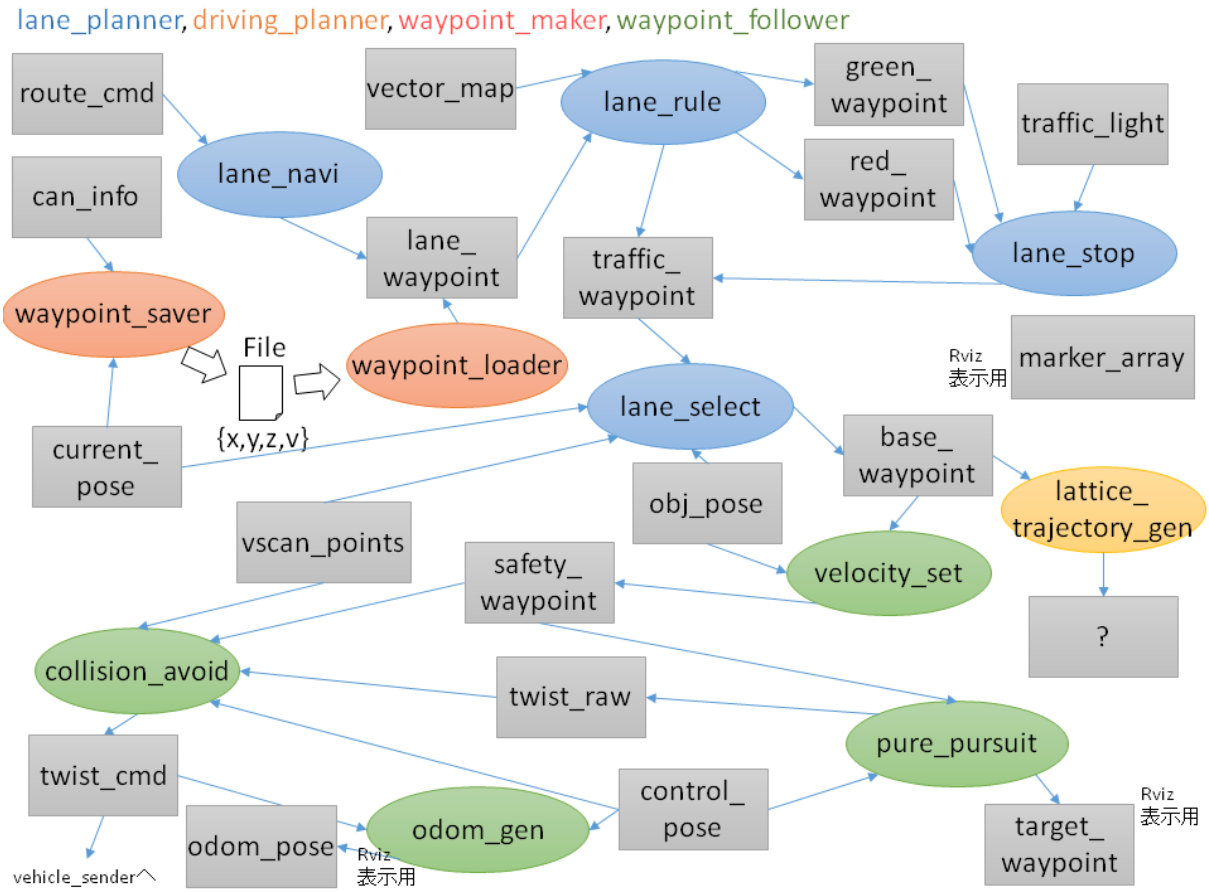


図 6 経路探索

3 次元地図などのデータの読み込み(DB、ファイル)

ros/src/data

map_db, map_file, pos_db, dynamic_map

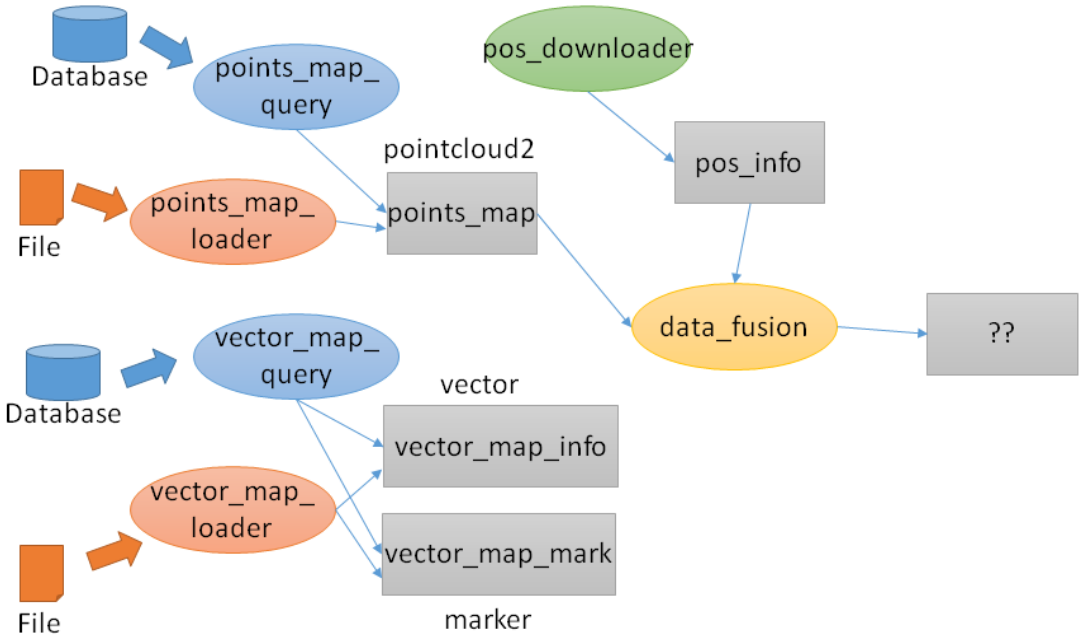


図 7 3 次元地図などのデータの読み込み

ドライバおよびセンサフュージョン

ros/src/sensing/drivers および ros/src/sensing/fusion

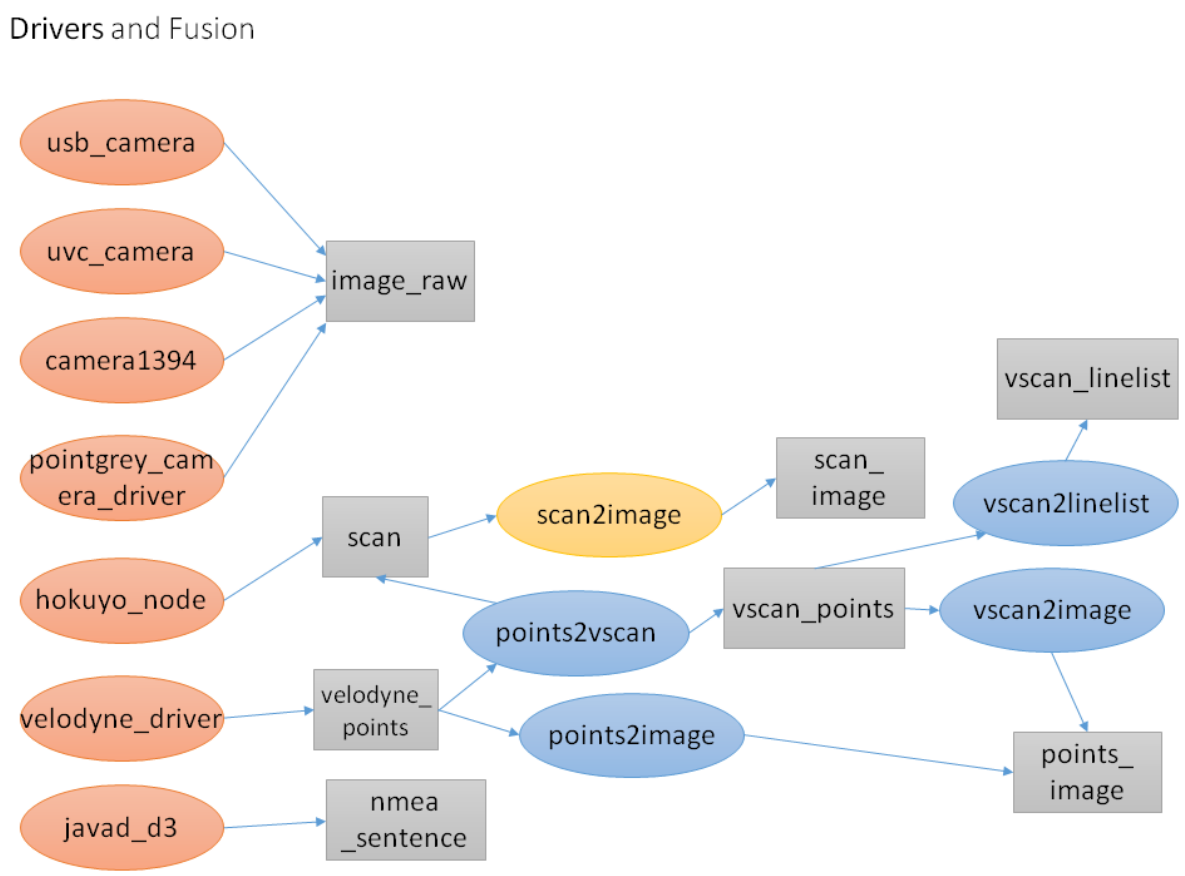


図 8 ドライバおよびセンサフュージョン

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/socket

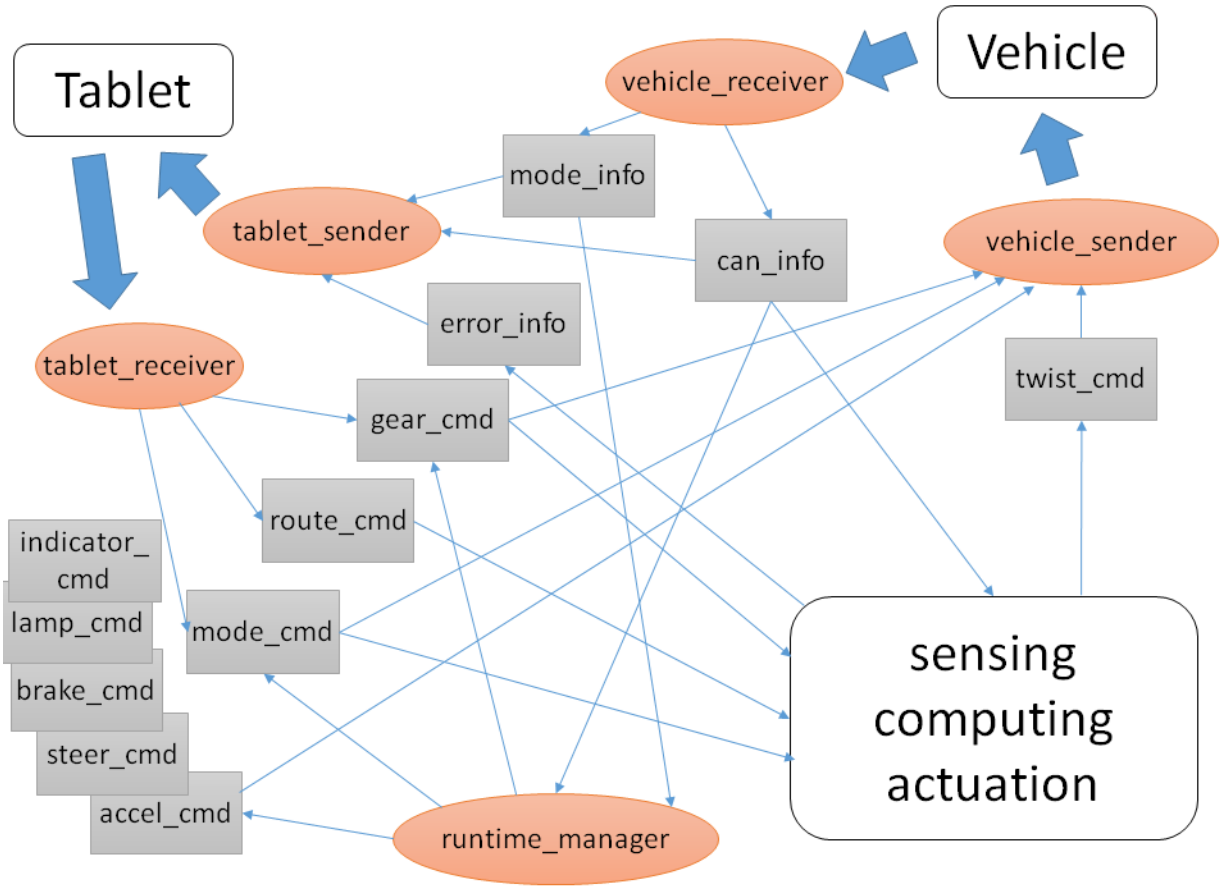


図 9 スマートフォン用アプリケーションとのインタフェース

その他

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

チャプター
3

3. ノードの作成

ここではAutoware で利用可能なノードを作成するためのおおまかな手順を示します。

開発の流れ

基本的には ROS のノード作成の手順と同じです(ROS のチュートリアルが参考になります)。

http://wiki.ros.org/ja/ROS/Tutorials (日本語)

http://wiki.ros.org/ROS/Tutorials (英語)

開発の流れは以下のとおりです。

4. Package を作成する
   1. ノードを作成する
   2. ノードをビルドする
   3. ノードの動作確認をする

パッケージを作成する

1. ROS の場合、以下の条件を満たしたものが、パッケージとみなされます。
   - ✓ パッケージのメタ情報が記述された設定ファイル package.xml を直下に含む。
   - ✓ ビルドシステム CMake(http://www.cmake.org/)の設定ファイル CMakeLists.txt を直下に含む。
   - ✓ 1 つのディレクトリに 1 つの package のみ存在する (入れ子になっていない)。

Autoware では、上記に加え、以下の条件を満たすことが推奨されています。

- ✓ ros/src/sensing/fusion などの各カテゴリに packages ディレクトリがあり、その直下にパッケージを作成する。
- ✓ nodes ディレクトリにノード名と同じディレクトリを作成し、その下にノードのソースコードなどを配置する。
- ✓ msg ディレクトリにメッセージファイルを配置する。
- ✓ ノード等で共通の処理をライブラリ化している場合は、lib や include などのディレクトリに、ライブラリのソースコードを配置する。

注: 現在は、1 つのノードをネームスペースによって使い分ける方法が推奨されています。

上記をまとめると、以下のようになります。

```
ros/src/カテゴリ..../packages/パッケージ名/
        package.xml
        CMakeLists.txt
        nodes/
                ノード名 1/
                ノード名 1.cpp
                ノード名 1.py            その他ソースコード
        ノード名 2/
                ノード名 2.cpp
                ノード名 2.py            その他ソースコード
    …
        msg/
                メッセージ名 1.msg
                メッセージ名 1.msg
        …
        include/
        ライブラリ等のヘッダファイル…
        lib/
        ライブラリのソースコード
```

2. パッケージを作成するには、catkin_create_pkg コマンドを使用します。

catkin_create_pkg パッケージ名 依存するパッケージ名...

Autoware では、packages ディレクトリに移動して、以下のように実行します。

```
$ cd ros/src/カテゴリ/packags/
$ catkin_create_pkg mypkg roscpp std_msgs
Successfully created files in /foo/ros/src/bar/packages/mypkg.
Please adjust the values in package.xml.
$ ls mypkg/
CMakeLists.txt        include/  package.xml  src/
```

CMakeLists.txt および package.xml を、適宜修正します。

CMakeLists.txt に関しては、ビルドの手順で述べます。

package.xml に関しては、maintainer や license、description、version などを適切な値に変更してください。

3. rospack コマンドで、パッケージに関する情報を確認できます。第一引数に find、第二引数にパッケージ名を指定して実行すると、パッケージの格納場所を確認できます。

```
$ rospack find roscpp
/opt/ros/indigo/share/roscpp
$ rospack find foo
[rospack] Error: package 'foo' not found
```

4. Packageの依存関係は、depends1(直接依存)やdepends(間接依存)で確認できます。

```
$ rospack depends1 mypkg
roscpp
std_msgs
$ rospack depends mypkg
cpp_common
rostime
roscpp_traits
roscpp_serialization
…
```

ノードを作成する

ここでは、簡単なノードとして、simplenode を C++で記述する例を示します。

Python による記述例は、前述の ROS のチュートリアルに記載されています。

まず、使用しない include と src ディレクトリを削除し、新たに nodes/simplenode ディレクトリを作成します。

```
$ pwd
/foo/ros
$ pushd src/bar/packages/mypkg/
$ rm -rinclude
$ rmdir src
$ mkdir -p nodes/mynode
```

次に、ノードのソースコード nodes/mynode/mynode.cpp を記述します。
以下に例を示します。

```
#include "std_msgs/String.h"
#include "std_msgs/Int32.h"

 static ros::Publisher pub;

 // 購読したトピックのコールバック関数
 static void sub_callback(const std_msgs::String& smsg)
 {
        std_msgs::Int32 pmsg;

        // メッセージの作成
        pmsg.data = smsg.data.size();

        // トピックの配信
        pub.publish(pmsg);
 }

 int main(int argc, char *argv[])
 {
        // ROS の初期化
        ros::init(argc, argv, "mynode");
        ros::NodeHandle n;

        // 配信するトピックの設定
        pub = n.advertise<std_msgs::Int32>("mypubval", 10);

        // 購読するトピックの設定
        ros::Subscriber sub = n.subscribe("mysubstr", 10, sub_callback);

        // メインループ
        ros::spin();

        return 0;
 }
```

mysubstr という文字列型(std_msgs::String)の topic を購読し、その文字数(std_msgs::Int32)を mypubval というトピックで配信する、単純なノードです。

ノードをビルドする

1. ビルドする前に、CMakeLists.txt を修正します。

まず、Autoware では、C++のコンパイル時に特定のオプションを指定しています。そのため、CMakeLists.txt に以下の 1 行を追加します。

set(CMAKE_CXX_FLAGS "-std=c++0x -O2 -Wall ${CMAKE_CXX_FLAGS}")

2. mynode のソースコードが nodes/mynode/mynode.cpp であることを示す、以下の 1 行を追加します。

add_executable(mynode nodes/mynode/mynode.cpp)

3. ROS の一般的なライブラリをリンク時に使用するよう、以下の 1 行を追加します。

```
target_link_libraries(mynode
  ${catkin_LIBRARIES}
)
```

4. トップディレクトリに戻って、catkin_make_release スクリプトを実行します。

```
$ popd
$ pwd
/foo/ros
$ ./catkin_make_release
```

ただし、catkin_make_release スクリプトは、すでにビルド済の実行ファイルやライブラリをすべて削除し、最初からビルドを実行し直します。毎回このスクリプトを実行すると時間がかかるため、通常は catkin_make コマンドでビルドを行います。

$ catkin_make -DCMAKE_BUILD_TYPE=Release

ノードの動作確認をする

1. まず、ROS の基本的なプログラム(Master や Parameter Server など)を起動するため、roscore コマンドを起動します。あるいは、./run の起動でも構いません。

```
$ . devel/setup.bash
$ roscore
```

2. 次に、作成した mynode を rosrun コマンドで起動します。引数は、パッケージ名とノード名です。
先の roscore 実行により、その擬似端末は roscore に専有されるため、別の擬似端末から実行します。

```
$ . devel/setup.bash
$ rosrun mypkg mynode
```

3. 引数に list を指定して rosnode コマンドを実行すると、実行中のノードを確認できます。

```
$ rosnode list
/mynode
/rosout
```

4. 同様に、引数に list を指定して rostopic コマンドを実行すると、配信もしくは購読されているトピックの一覧を確認できます。

```
$ rostopic list
/mypubval
/mysubstr
/rosout
/rosout_agg
```

5. 本来は他のノードとトピックを送受信しますが、ここでは代わりにコマンドを使用します。
mynode は mypubval トピックを配信するため、引数に echo を指定して rostopic コマンドを実行することで、トピックを購読します。

```
$ rostopic echo /mypubval
```

6. トピックの配信にも rostopic コマンドを使用します。引数には、pub と、配信するトピック名、トピックの型、そして値を指定します。

```
$ rostopic pub -1 /mysubstr std_msgs/String "hello world"
publishing and latching message for 3.0 seconds
$
```

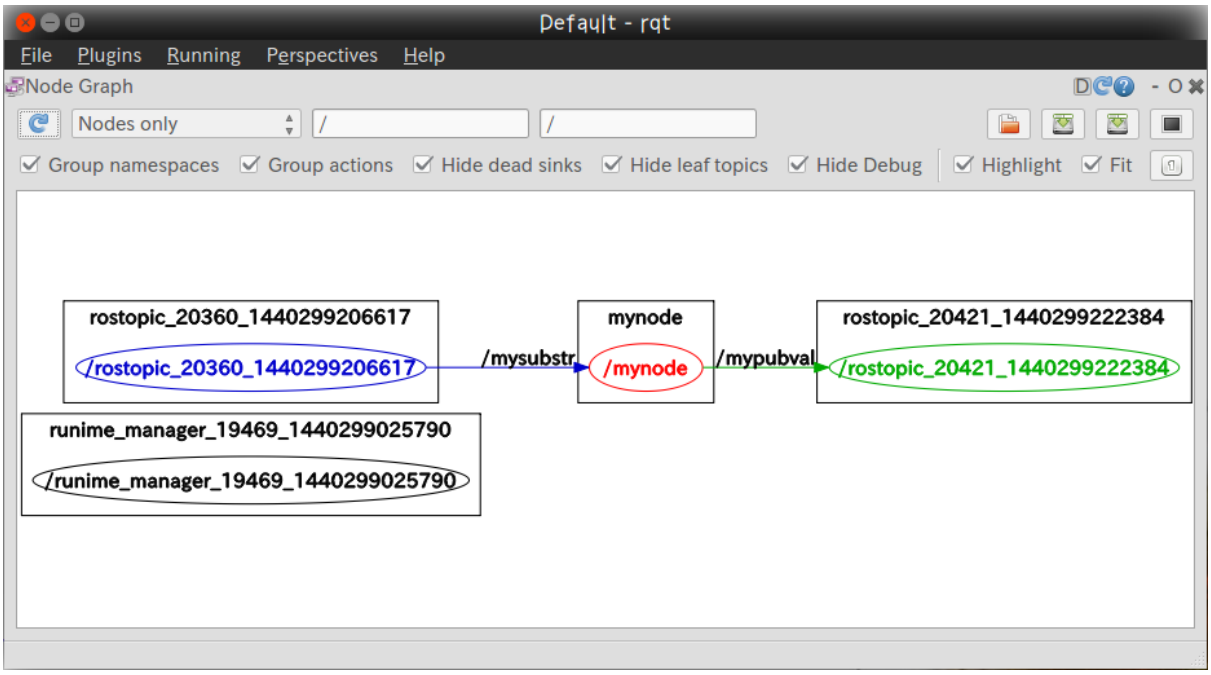
7. mynode が”hello world”を購読し、文字数を mypubval トピックに配信します。よって、先ほどの rostopic echo が以下を出力します。

```
$ rostopic echo /mypubval
data: 11
---
```

8. また、rqt_graph を使うと、ノード間がどのトピックでつながっているかを、グラフで確認できます。

```
$ rosrun rqt_graph rqt_graph
```

9. Runtime Manager の RQT ボタンをクリックすると、rqt が実行されますので、rqt 上で Plugins→Introspection→Node Graph を選択すると、rqt_graph が表示されます。



実際のノードの開発で、トピックによる購読や配信がうまく行われない場合は、rostopic コマンドや rqt_graph で、トピックのつながりなどを確認してください。

Runtime Manager

Runtime Manager から起動・終了する ROS ノードを追加する方法、起動する ROS ノードへ与えるパラメータを設定する方法を示します。

Computing タブから起動・終了する ROS ノードの追加例

Computing タブに表示される各欄の項目は、次のパスの設定ファイルに記述されています。

ros/src/util/packages/runtime_manager/scripts/computing.yaml

例えば、Localization/ndt_localizer 欄の ndt_matching 項目の設定は、設定ファイル中の次の箇所に記述されています。

```
name : Computing
subs :
 :
<略>
 :
    - name : ndt_localizer
      subs :
 :
<略>
 :
    - name : ndt_matching
        cmd  : roslaunch ndt_localizer ndt_matching.launch
        param: ndt
```

1. ndt_matching 項目のチェックボックスを ON にすると、サブプロセスを起動し、cmd 行に記述されたコマンド"roslaunch ndt_localizer ndt_matching.launch"を実行し、ndt_localizer パッケージの ndt_matching.launch スクリプトを起動します。チェックボックスを OFF にすると、起動しているサブプロセスを終了し、起動している ndt_maching.luanch スクリプトを終了させます。

2. Motion Planning 欄直下の階層の末尾に、新たに Example 欄を追加し、そこに TurtleSim 項目を追加して、turtlesim パッケージの turtlesim_node ノードを起動・終了させる場合の設定の追加例を示します。

```
name : Computing
 subs :
  :
 <略>
  :
  - name : Motion Planning
    subs :
      - name : driver_lanner
        subs :
        - name : obstacle_avoidance
          cmd  : rosrun driving_lanner obstacle_avlidance
  :
 <略>
  :
        - name : car_simulation
          cmd  : roslaunch waypoint_follower car_simulator.launch
          param: car_simulator
          gui  :
  :
 <略>
  :
              yaw:
                depend      : use_pose
                depend_bool : 'lambda v : v == "Initial Pos"'
                flags : [ no_category, nl ]
        - name : Example              # この行を追加
          subs :                                # この行を追加
          - name : TurtleSim          # この行を追加
            cmd  : rosrun turtlesim turtlesim_node # この行を追加
```

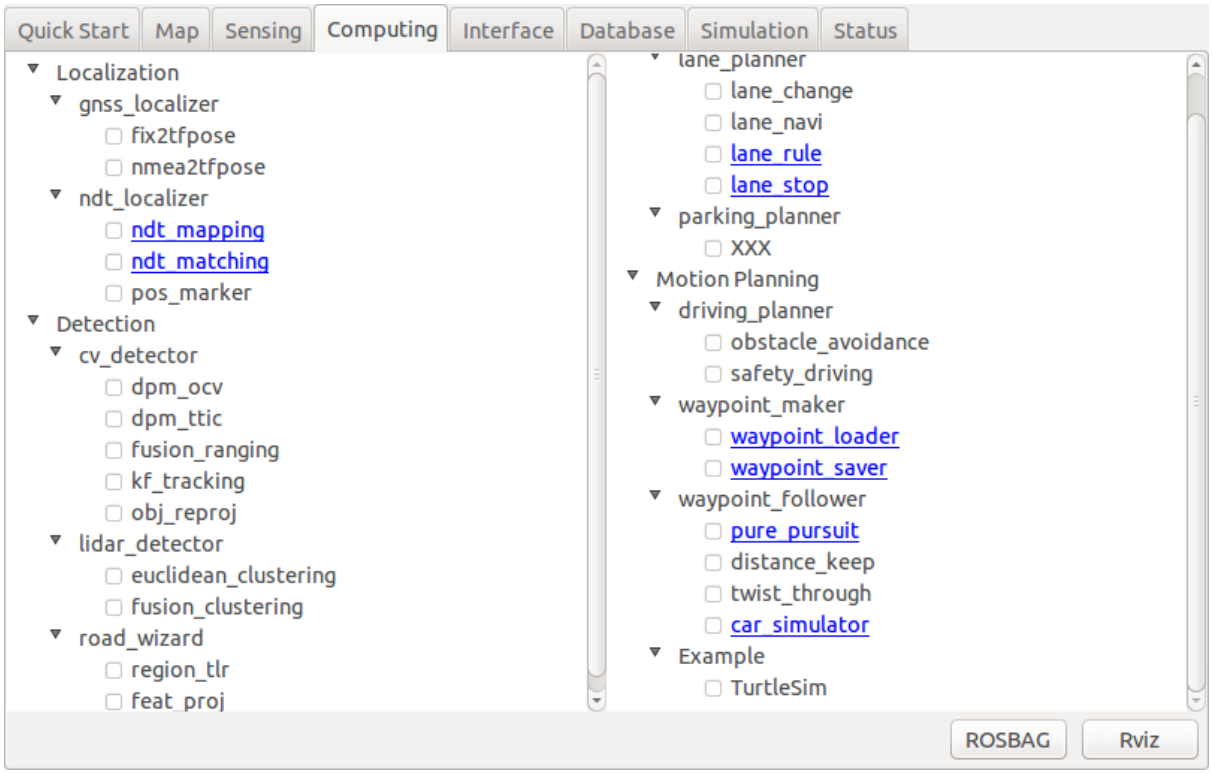


図 10 Comptuing タブ追加項目の表示

Computing タブから ROS ノードへ与えるパラメータの設定例

例えば、Localization/ndt_localizer 欄 ndt_matching 項目は、リンクが設定された状態で表示され、項目をクリックすると、パラメータを調整するダイアログが表示されます。

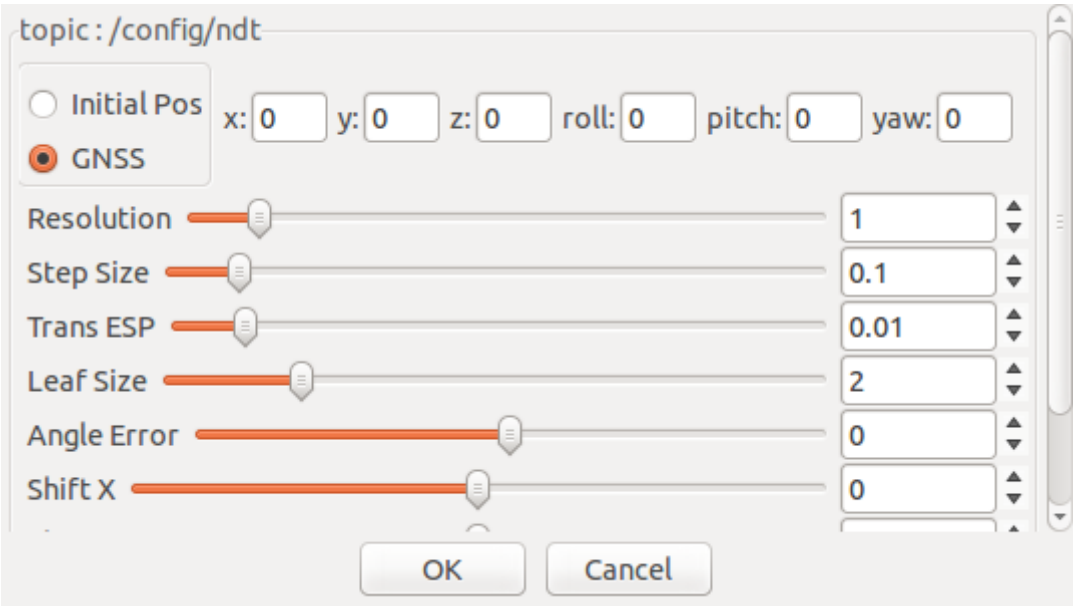


図 11 パラメータを調整するダイアログ

この例では、パラメータの値を変更すると、パラメータはトピック /config/ndt として発行され、ndt_matching.launch スクリプトから起動しているノードで購読されます。

ダイアログに表示されるパラメータは、次のパスの設定ファイルに記述されています。

ros/src/util/packages/runtime_manager/scripts/computing.yaml

Localization/ndt_localizer 欄の ndt_matching 項目の設定は、設定ファイル中の次の箇所に記述されています。

```
name : Computing
subs :
 :
<略>
 :
    - name : ndt_localizer
      subs :
 :
<略>
 :
    - name : ndt_matching
         cmd  : roslaunch ndt_localizer ndt_matching.launch
         param: ndt
```

param 行の ndt の記述は、パラメータ名が ndt であり、ダイアログに表示するパラメータの詳細が、後方の params 行以降にある "name : ndt" に記述されている事を表しています。

```
params :
 :
<略>
 :
 - name  : ndt
   topic : /config/ndt
   msg   : ConfigNdt
   vars  :
   - name  : init_pos_gnss
     kind  : radio_box
     choices:
     - Initial Pos
     - GNSS
     v     : 1
   - name  : x
     label : 'x:'
     v     : 0.0
   - name  : y
     label : 'y:'
     v     : 0.0
   - name  : z
     label : 'z:'
     v     : 0.0
   - name  : roll
     label : 'roll:'
     v     : 0.0
   - name  : pitch
     label : 'pitch:'
     v     : 0.0
   - name  : yaw
     label : 'yaw:'
     v     : 0.0
 :
<略>
 :
   - name  : shift_y
     label : Shift Y
     min   : -2.0
     max   : 2.0
     v     : 0
   - name  : shift_z
     label : Shift Z
     min   : -2.0
     max   : 2.0
     v     : 0
```

この設定例では、topic 行に発行するトピック名、msg 行にトピックで使用するメッセージ型名、vars行以下に、メッセージに含まれる各パラメータの設定が記述されています。

vars 行以下の各パラメータの設定では、name 行にメッセージ型のメンバ名、label 行にダイアログで表示するラベル文字列、min 行にパラメータの最小値、max 行にパラメータの最大値、v 行にパラメータの初期値が記述されています。

パラメータ追加例

Motion Planning 欄直下の階層の末尾に、新たに Example 欄を追加し、そこに TurtleSim 項目を追加した後、Int32 型のパラメータを追加して、メッセージのパラメータをトピックとして発行する設定例を示します。

1. まず、設定ファイルに TrutleSim 項目を追加します。

```
name : Computing
subs :
 :
<略>
 :
 - name : Motion Planning
   subs :
     - name : driver_lanner
       subs :
       - name : obstacle_avoidance
         cmd  : rosrun driving_lanner obstacle_avlidance
 :
<略>
 :
       - name : car_simulation
         cmd  : roslaunch waypoint_follower car_simulator.launch
         param: car_simulator
         gui  :
 :
<略>
 :
             yaw:
               depend      : use_pose
               depend_bool : 'lambda v : v == "Initial Pos"'
               flags : [ no_category, nl ]
       - name : Example                        # この行を追加
         subs :                              # この行を追加
         - name : TurtleSim                      # この行を追加
           cmd  : rosrun turtlesim turtlesim_node  # この行を追加
```

2. 次に、パラメータ名 example_param を指定する param 行を追加します。

```
    - name : Example
      subs :
      - name : TurtleSim
        cmd  : rosrun turtlesim turtlesim_node
        param: example_param                # この行を追加
```

3. さらに、後方の params 行以降に、example_param の詳細設定を追加します。

```
params :
 :
<略>
 :
   - name  : dispersion
     label : Coefficient of Variation
     min   : 0.0
     max   : 5.0
     v     : 1.0

  - name  : example_param    # この行を追加
    topic : /example_topic   # この行を追加
    msg   : Int32            # この行を追加
    vars  :                  # この行を追加
    - name  : data           # この行を追加
      label : Parameter      # この行を追加
      min   : 0              # この行を追加
      max   : 100            # この行を追加
      v     : 50             # この行を追加
```

この例では、トピック名を /example、メッセージ型を Int32、メッセージ型 Int32 に含まれるメンバ data について、ダイアログに表示するラベル文字列を 'Parameter'、最小値を 0、最大値を 100、初期値を 50 に設定しています。

4. メッセージ型 Int32 は、Runtime Manger で使用してない型なので、Runtime Mananger の Python スクリプト
(ros/src/util/packages/runtime_manager/scripts/runtime_manager_dialog.py)
の冒頭の include 行の箇所に、メッセージ型 Int32 の include 行を追加します。

```
  :
 <略>
  :
from runtime_manager.msg import accel_cmd
from runtime_manager.msg import steer_cmd
from runtime_manager.msg import brake_cmd
from runtime_manager.msg import traffic_light
from std_msgs.msg import Int32                # この行を追加

class MyFrame(rtmgr.MyFrame):
  :
 <略>
  :
```

5. Runtime Manger を起動すると、Computing タブに追加した項目が、リンク設定された状態で表示されます。

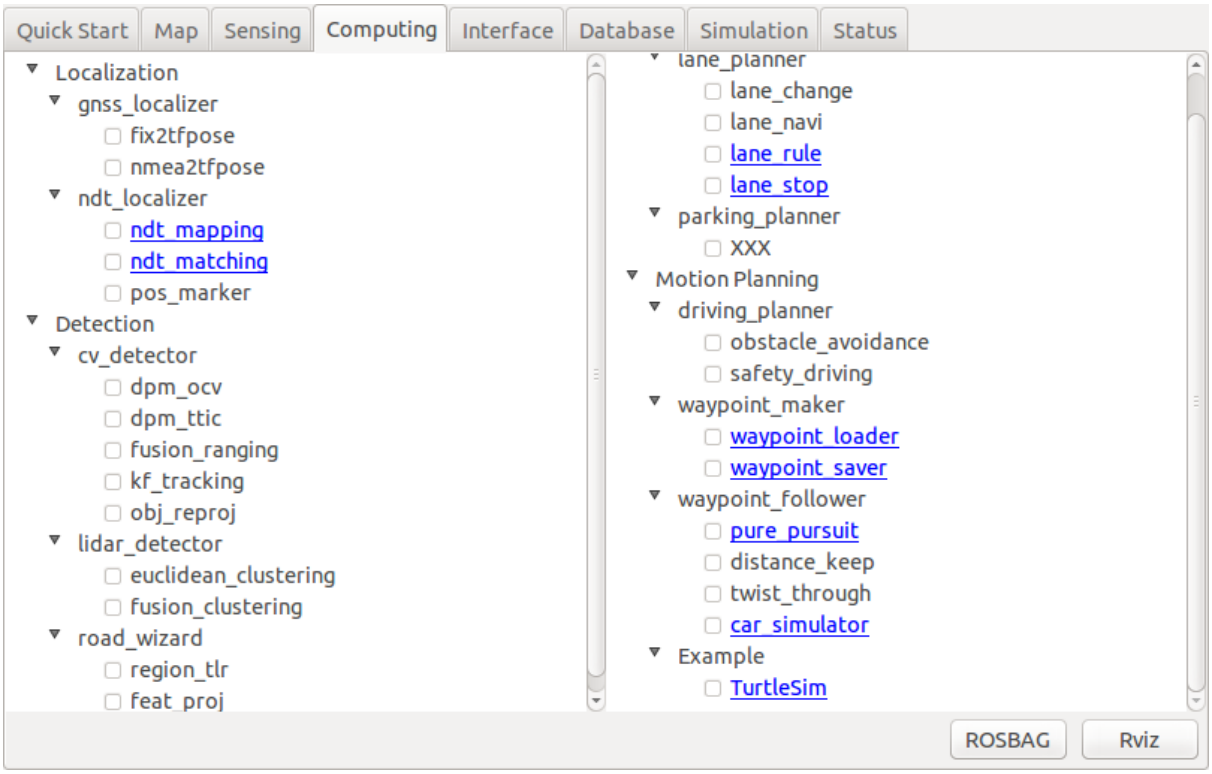


図 12 Computing タブ追加項目のリンク設定表示

6. 項目をクリックするとダイアログが表示されます。

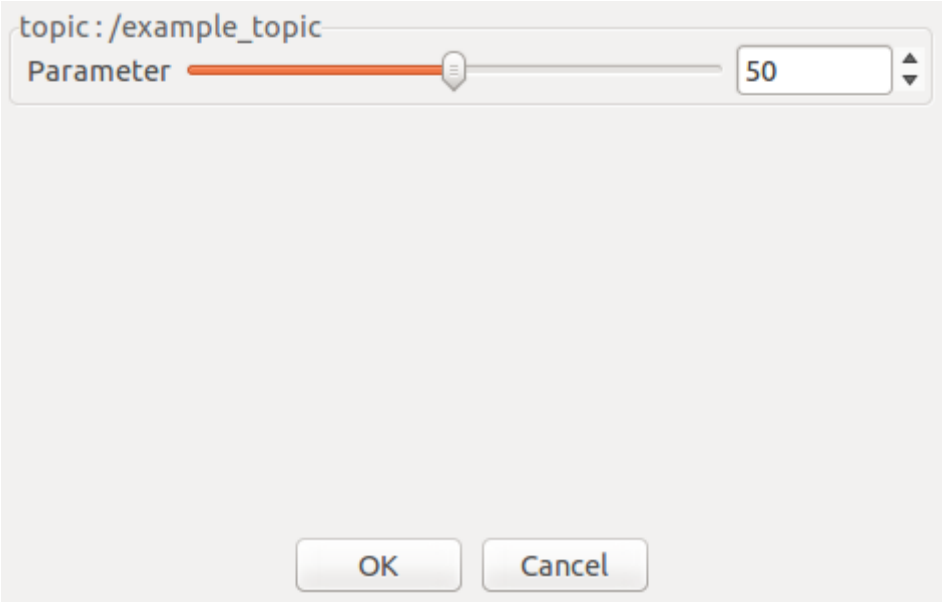


図 13 追加項目のパラメータ設定ダイアログ

7. トピックを表示するため、別端末で次のコマンドを実行します。

$ rostopic echo /example_topic

8. ダイアログでパラメータを変更すると、発行トピックの内容が表示されます。

data: 51
---
data: 52
---
data: 53

小数値のパラメータおよびスライダー表示

パラメータ設定の min 行、max 行、v 行のいずれかが小数点を含む値の場合は、小数値のパラメータと解釈されます。

また、パラメータの設定に min 行、max 行の指定が無い場合は、最大値、最小値が判らないためスライダーは表示されません。

小数値のパラメータおよびスライダー表示

リンク設定からパラメータ設定ダイアログを開き、ファイルパス文字列を設定して、そのパス文字列を rosparam パラメータとして設定する例を示します。

1. 先の例で追加した Example 欄 TrutleSim 項目のパラメータ example_param に、ファイルパスの設定項目を追加します。

2. 設定ファイル computing.yaml にファイルパスの設定を追加します。

```
:
<略>
 :
  - name  : example_param
    topic : /example_topic
    msg   : Int32
    vars  :
    - name  : data
      label : Parameter
      min   : 0
      max   : 100
      v     : 50
    - name     : data_file_path            # この行を追加
      kind     : path                      # この行を追加
      v        : /tmp/foo                  # この行を追加
      rosparam : /example_param/data_path_1   # この行を追加
```

3. name 行は、トピックとしてメッセージ出力する場合は、メッセージ中のメンバ名を指定します。ここでは、ファイルパス文字列は、トピックのメッセージ中に存在せず、パス文字列を rosparam パラメータとして設定する例を示します。この場合、name 行の指定は vars 内の識別用として、他と重複しない任意の名前を指定すればよいです。

4. kind 行は、ファイルパス文字列を表す"path"を指定します。

5. v 行は、デフォルト値のパスを指定します。

6. rosparam 行は、rosparam パラメータの名前を指定します。

7. Runtime Manager を起動し、Computing タブの TrutleSim 項目のリンクをクリックするとダイアログが表示されます。

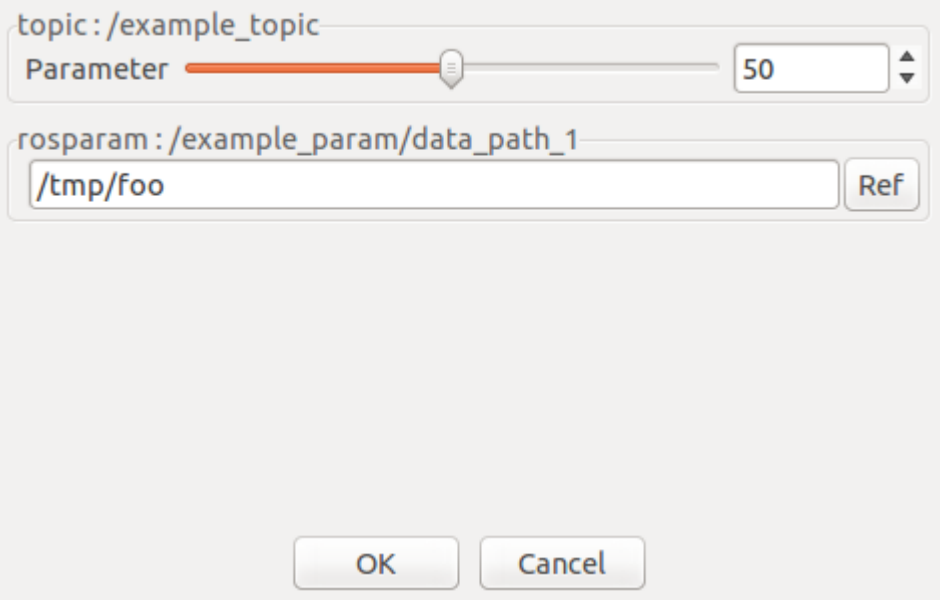


図 14 ファイルパス設定を追加したパラメータ設定ダイアログ

8. Ref ボタンからファイルを選択したり、テキストボックスにパスを入力し ENTER キーで設定すると、指定の rosparam パラメータに設定した値がセットされます。

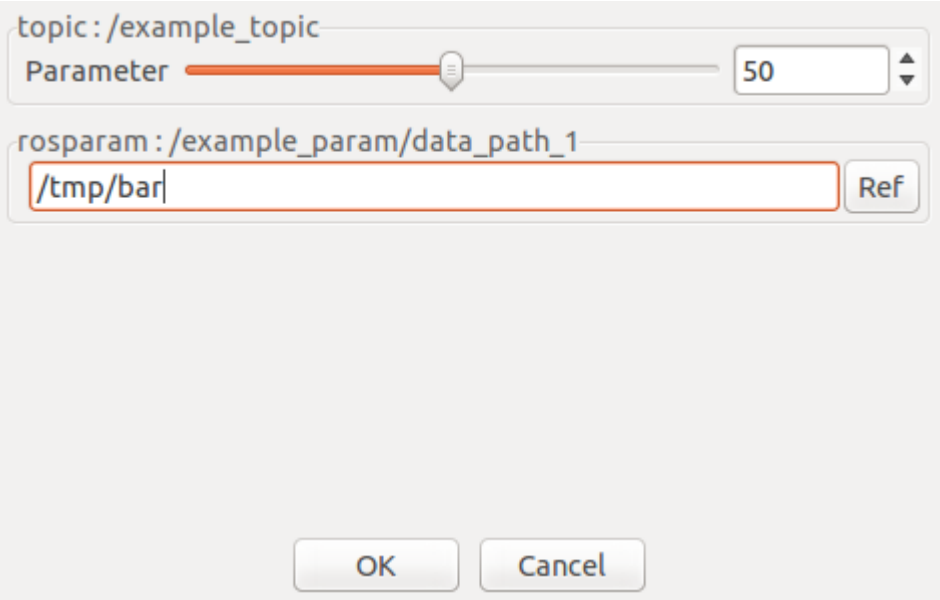


図 15 パラメータ設定ダイアログでパスを入力

9. rosparam パラメータを表示するため、別端末で次のコマンドを実行します。

```
$ rosparam get /example_param/data_path_1
/tmp/bar
$
```

10. Cancel ボタンでダイアログを閉じた場合は、指定の rosparam パラメータの値が、ダイアログでを開いた時点の値に戻されます。

11. Runtime Manager を終了すると、ダイアログで設定したパラメータの値は、パラメータ保存ファイルに保存されます。

パラメータ保存ファイル ros/src/util/packages/runtime_manager/scripts/param.yaml

```
 :
<略>
 :
TurtleSim:
  data: 50
  data_file_path: /tmp/bar
 :
<略>
 :
```

ディレクトリを選択する場合

Ref ボタンでファイルではなく、ディレクトリを選択したい場合は、パラメータ data_file_path の設定に、path_type 行で dir 指定を追加します。

設定ファイル computing_launch_cmd.yaml

```
:
<略>
 :
  - name  : example_param
    topic : /example_topic
    msg   : Int32
    vars  :
    - name  : data
      label : Parameter
      min   : 0
      max   : 100
      v     : 50
    - name     : data_file_path
      kind     : path
      path_type: dir                          # この行を追加
      v        : /tmp                      # 適宜変更
      rosparam : /example_param/data_path_1
```

Runtime Manager を再起動する際は、パラメータ保存ファイル中に保存されている、パス文字列の設定を削除してから起動します。

```
$ cd ros/src/util/packages/runtime_manager/scripts
$ cp param.yaml param.yaml-
$ sed -e '/data_file_path:/d' param.yaml- > param.yaml
```

Runtime Manager を起動し、Computing タブの TrutleSim 項目のリンクをクリックするとダイアログが表示されます。

Ref ボタンでディレクトリを選択するダイアログが表示されるようになります。

パラメータをコマンドライン引数として出力する場合

設定したファイルパス文字列を、チェックボックスで起動するコマンドの、コマンドライン引数として与えたい場合の設定例を示します。

1. 確認のため、TurtleSim 項目の実行コマンドとして設定している文字列を、"echo" に変更しておきます。

設定ファイル computing.yaml

```
:
<略>
:
      - name : Example
        subs :
        - name : TurtleSim
          #cmd  : rosrun turtlesim turtlesim_node  # 変更
          cmd  : echo                          # 変更
          param: example_param
```

2. パラメータ example_param に、新たなファイルパス設定を追加し、コマンドライン引数として出力するよう設定します。

設定ファイル computing_launch_cmd.yaml

```
:
<略>
:
 - name  : example_param
   topic : /example_topic
   msg   : Int32
   vars  :
   - name  : data
     label : Parameter
     min   : 0
     max   : 100
     v     : 50
   - name    : data_file_path
     kind    : path
     path_type: dir
     v       : /tmp
     rosparam : /example_param/data_path_1
   - name    : data_file_path_2       # 追加
     kind    : path                           # 追加
     v       : /tmp/bar                 # 追加
     cmd_param:                                    # 追加
       delim   : ''                             # 追加
```

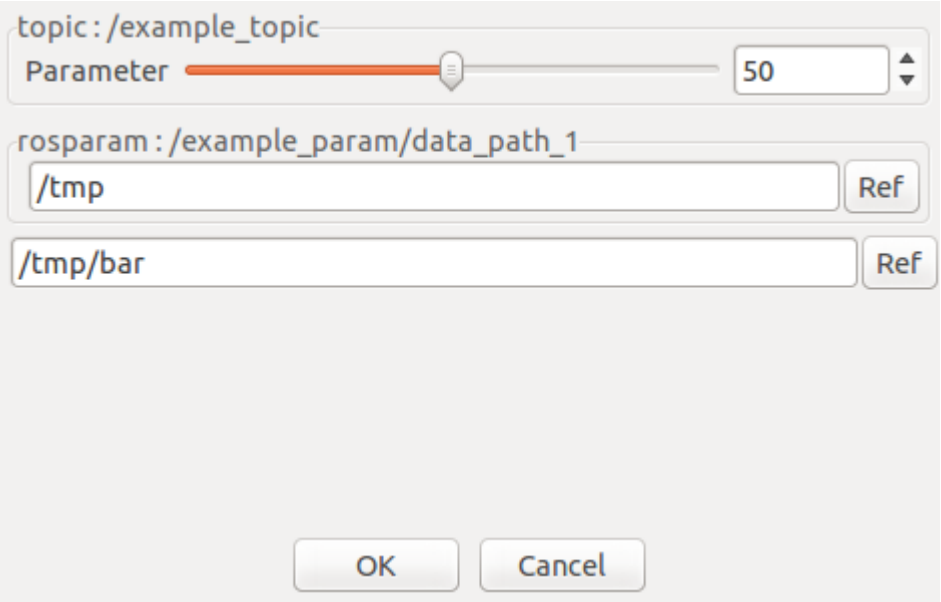


図 16 パラメータ設定ダイアログ

3. OK ボタンでダイアログを閉じ、TurtleSim 項目のチェックボックスを ON にすると、Runtime Manager を起動した端末に、次の表示が出ます。

['echo', '/tmp/bar']
/tmp/bar

4. cmd 行に設定した echo コマンドの引数として、ファイルパスが指定されて実行されます。

cmd_param 行の設定

cmd_param 行の設定として、dash 行、var_name 行、delim 行を指定することができます。

実行コマンドとコマンドライン引数は、次の並びに配置されます。

<cmd 行の値><空白><dash 行の値><var_name 行の値><delim 行の値><パラメータの値>

dash 行が存在しない場合は、<dash 行の値><var_name 行の値> の部分を出力しません。

delim 行が存在しない場合は、<delim 行の値><パラメータの値> の部分を出力しません。

var_name 行が存在しない場合は、デフォルトとして name 行の値が使われます。

dash 行で ''(空文字列)を指定した場合は、<dash 行の値><var_name 行の値> の部分は <var_name 行の値> だけになります。
delim 行で ''(空文字列)を指定した場合は、<delim 行の値><パラメータの値> の部分は <パラメータの値> だけになります。

設定例

```
cmd_param:
 dash    : '--'
 delim   : '='

コマンドラインの配置
echo --data_file_path_2=/tmp/bar


cmd_param:
 dash    : '-'
 var_name : f
 delim   : ' '              # 1 つの空白文字

コマンドラインの配置
echo -f /tmp/bar


cmd_param:
 dash    : ''       # 空文字
 delim   : ':='

コマンドラインの配置
echo data_file_path_2:=/tmp/bar


cmd_param:
 delim   : ''       # 空文字

コマンドラインの配置
echo /tmp/bar


cmd_param:
 dash    : '--'
 var_name : ''       # 空文字
 delim   : ''       # 空文字

コマンドラインの配置
echo --/tmp/bar
```

パラメータ設定のその他の kind 行指定

kind 行でチェックボックス、トグルボタン、ラジオボックス、メニューを指定できます。

チェックボックス、トグルボタンでは BOOL 値(True/False)を扱い、ラジオボックス(複数のラジオボタンをまとめた部品)、メニューでは選択されている項目のインデックス値(0 から項目数-1 までの整数値)を扱います。

設定ファイル computing.yaml

```
:
<略>
:

 - name  : example_param
   topic : /example_topic
   msg   : Int32
   vars  :
   - name  : data
     label : Parameter
     min   : 0
     max   : 100
     v     : 500
   - name    : data_file_path
     kind    : path
     path_type: dir
     v       : /tmp
     rosparam : /example_param/data_path_1
   - name    : data_file_path_2
     kind    : path
     v       : /tmp/bar
     cmd_param:
       delim   : ''
   - name    : sw_1                    # 追加
     label   : Enable                  # 追加
     kind    : checkbox                # 追加
     v       : True                    # 追加
   - name    : sw_2                    # 追加
     label   : Alert                   # 追加
     kind    : toggle_button           # 追加
     v       : False                   # 追加
   - name    : sel_1                   # 追加
     kind    : radio_box               # 追加
     label   : 'Edit:'                 # 追加
     choices : [ cut, copy, paste ]    # 追加
     v       : 1                       # 追加
   - name    : sel_2                   # 追加
     kind    : menu                    # 追加
     choices : [ open, close, save, load ]    # 追加
     v       : 2                       # 追加
```

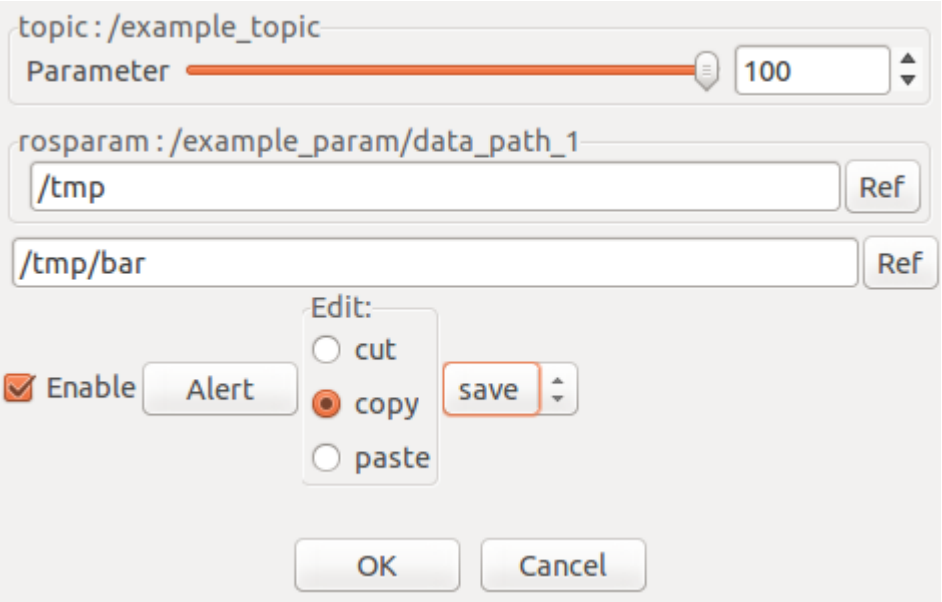


図 17 部品が追加されたパラメータ設定ダイアログ

この例では、ダイアログにパラメータの部品が追加されるだけで、パラメータの値は出力されません。追加したパラメータを出力するには、以下の 3 つの方法があります。

- ✓ name 行を、トピックのメッセージに含まれるメンバ名に設定すれば、トピックとして出力される。
- ✓ rosparam 行で rosparam パラメータ名を指定すれば、rosparam パラメータとしてセットされる。
- ✓ cmd_param 行を指定すれば、項目のチェックボックス ON でコマンドを起動する際に、コマンド引数として指定される。

Quick Start タブのボタンで起動・終了するコマンドの設定例

Quick Start タブの最下行のトグルボタンで起動・終了するコマンドは、次のパスの設定ファイルに記述されています。

ros/src/util/packages/runtime_manager/scripts/qs.yaml

例えば、Sensing トグルボタンの設定は、設定ファイル中の次の箇所に記述されていま

す。

```
buttons:
 :
<略>
 :
 sensing_qs:
  run   : roslaunch
  param : sensing_qs
 :
<略>
 :
```

Sensing トグルボタンを ON にすると、サブプロセスを起動し、run 行に記述されたコマンド"roslunch"を param 行で指定された内容のコマンドライン引数(.launch ファイル)を与えて実行します。

トグルボタンを OFF にすると、起動しているサブプロセスを終了し、実行しているコマンド(roslaunch コマンド)を終了させます。

param 行の sensing_qs の記述は、パラメータ名が sensing_sq であり、コマンドライン引数として追加するパラメータの詳細が、後方の params 行以降にある "name : sensing_sq"に記述されている事を表します。

```
params :
 :
<略>
 :
 - name  : sensing_sq
   vars  :
   - name  : file
     kind  : path
     v     : ''
     cmd_param :
       delim     : ''
       must      : True
 :
<略>
 :
```

この設定例では、vars 行以下にコマンドライン引数として追加するパラメータの設定が記述されています。

vars 行以下のパラメータ設定では、name 行で vars 内の識別用の名前として file を指定し、kind 行で、ファイルパス文字列を表す"path"を指定し、v 行で、値のパスとして""(空文字列)を指定し、cmd_param 行以下の設定で、上記のパス文字列をコマンドライン引数として指定する際の形式を指定しています。

この設定例の形式では、v 行の値のパス文字列のみをコマンドライン引数として与えるように指定しています。

cmd_param 行の設定の詳細は「パラメータをコマンドライン引数として出力する場合」「cmd_param 行の設定」を参照してください。

また、kind 行として"patth"が指定されている場合は、Runtime Manager スクリプト内で、v 行の値をパス文字列として扱い、絶対パスに変換してからコマンドライン引数として配置しています。

Senging テキストボックスにパス文字列”~/.autoware/launch_files/sensing.launch”を入力し、Sensing トグルボタンを ON にすると、次のコマンドが実行されます。

roslaunch (ユーザのホームディレクトリの絶対パス)/.autoware/launch_files/sensing.launch

例えば、設定ファイルの内容を次のように変更すると、実行するコマンドのコマンドライン引数を確認できます。

```
buttons:
 :
<略>
 :
 sensing_qs:
   #run   : roslaunch          # この行を変更
   run    : echo               # この行を追加
   param : sensing_qs
 :
<略>
 :


params :
 :
<略>
 :
 - name  : sensing_qs
   vars  :
   - name  : file
     kind  : path
     #v    : "# この行を変更
     v     : /tmp/foo                          # この行を追加
     cmd_param :
       dash     : '--'                  # この行を追加
       delim    : '='                          # この行を変更
       must     : True
   - name  : xval                               # この行以降を追加
     v     : 12.3
     cmd_param :
       dash     : '-'
       delim    : ' '
 :
<略>
 :
```

Runtime Manager終了後、パラメータ保存ファイルに保存されている変更前のパスを削除します。

パラメータ保存ファイル ros/src/util/packages/runtime_manager/scripts/param.yaml

```
 :
<略>
 :
sensing_qs:
  file: ~/.autoware/launch_files/sensor.launch
 :
<略>
 :
```

上記の sensor 行と file 行の 2 行を削除します。

Runtime Manager を起動しなおし、Sensing トグルボタンを ON にすると、Runtime Manger を起動した端末に次の表示がでます。

['echo', '--file=/tmp/foo', '-xval', '12.3']
--file=/tmp/foo -xval 12.3

Sensing タブのボタンで起動・終了するコマンドの設定例

Sensing タブ右側に配置されたトグルボタンで起動・終了するコマンドは、次のパスの設定ファイルに記述されています。

ros/src/util/packages/runtime_manager/scripts/sensing.yaml

例えば、Points Image トグルボタンの設定は、設定ファイル中の次の箇所に記述されています。

```
 :
<略>
 :
buttons:
 :
<略>
 :
 points_image :
   run   : rosrun points2image points2image
 :
<略>
 :
```

Points Image トグルボタンを ON にすると、サブプロセスを起動し、run 行に記述されたコマンド　"**rosrun points2image points2image"** を実行します。

トグルボタンを OFF にすると、起動しているサブプロセスを終了し、実行しているコマンド(rosrun コマンド)を終了させます。

例えば、設定ファイルの内容を次のように変更して、コマンド実行の様子を確認できます。

```
 :
<略>
 :
buttons:
 :
<略>
 :
 points_image :
   #run   : rosrun points2image points2image      # この行を変更
   run   : echo hello                              # この行を追加
 :
<略>
 :
```

Runtime Manager を起動し Sensing タブの画面を選択し、Points Image トグルボタンを ON にすると、Runtime Manger を起動した端末に次の表示がでます。

['echo', 'hello']
hello

//

チャプター
4

4. 環境構築

対応している OS や必要なソフトウェアについて説明します。

インストール

PC に以下の手順で OS(Linux)、ROS、Autoware などをインストールします。

OS

2015 年 9 月時点で 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

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" > \   /et
```

```
c/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-nmea-msgs \ ros-indig
o-sound-play
$ sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev
\ libssh2-1-dev libarmadillo-dev
```

~/.bashrc などに以下を追加します。

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

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。

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > \ /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-get install ros-hydro-desktop-full ros-indigo-nmea-msgs \ ros-hydro
-sound-play
$ sudo apt-get install libnlopt-dev freeglut3-dev libssh2-1-dev libarmadillo-dev
```

~/.bashrc などに以下を追加します。

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

Velodyne ドライバ

https://github.com/ros-drivers/velodyne 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。

```
$ sudo apt-get install libpcap-dev git
$ mkdir -p ~/ros_drivers/src
$ cd ~/ros_drivers/src
$ catkin_init_workspace
$ git clone https://github.com/ros-drivers/velodyne.git
$ cd ~/ros_drivers
$ catkin_make
$ source devel/setup.bash
```

OpenCV

OpenCV のサイト(http://sourceforge.net/projects/opencvlibrary/)から、バージョン 2.4.8 以降の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。

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

Qt（必要な場合）

Ubuntu14.04 の場合は qtbase5-dev および libqt5opengl5-dev パッケージはインストール済のため、下記の作業は必要ありません。

1. 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 git://code.qt.io/qt/qt5.git
   $ 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（必要な場合）

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に CUDA が必要となります。

インストールする場合は http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/ を参考に、以下の手順でインストールします。

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）

   $ 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_Utilities/deviceQuery/
   $ make
   $ ./deviceQuery

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

   export PATH=”/usr/local/cuda:$PATH”
   export LD_LIBRARY_PATH=”/usr/local/cuda/lib:$LD_LIBRARY_PATH”

FlyCapture（必要な場合）

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。

以下は2014年10月28日に試したときの手順です。
/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.
3. 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

4. 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

5. インストーラを起動します。

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

6. 以下が表示されるのでキーボードで「y」と入力します。

   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

以下が表示されるのでキーボードで「y」と入力します。

...
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

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。

github から最新を入手する場合

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

アーカイブを使用する場合

```
$ wget http://www.pdsl.jp/app/download/10394444574/Autoware-beta.zip
$ unzip Autoware-beta.zip
$ cd Autoware-beta/ros/src
$ catkin_init_workspace
$ cd ../
$ ./catkin_make_release
$ source devel/setup.bash
```

AutowareRider

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。

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

- 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、SSH を介してデータベースにアクセスします。その際、パスフレーズなしの SSH 鍵を使用します。

そのため、pos_db を使用する場合は、データベースサーバ用の SSH 鍵を以下の手順で作成し、SSH 公開鍵をデータベースサーバに登録する必要があります。

1. SSH 鍵の作成方法

   以下のコマンドを実行して鍵を作成します。

   $ ssh-keygen -t rsa

   その際、パスフレーズは空にして作成します。

   （文字列を入力せずに Enter キーを押下する）

   DSA を使用する場合は -t dsa と指定します。

2. SSH 公開鍵をデータベースサーバに登録する

   作成した SSH 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

   （”posup” はユーザ名、”db3.ertl.jp” はデータベースサーバ名）

   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。

チャプター
5

5. 用語

用語	解説
3次元地図	一般なカーナビゲーションなどで使われている2　次元の地図とは違い、道路脇辺に設置されている立体などを含めてさまざまな情報が取り込まれた地図。
Autoware	[Autoware]ROS上で動くオープンソースソフトウェア
AutowareRider	[Autoware]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Androidアプリケーション。TVドラマ「ナイトライダー」に似たUIを持つ。
AutowareRoute	[Autoware]MapFan SDK で実装された、経路データ生成のためのA　ndroidアプリケーション。
CAN	*Controller Area Network* 相互接続された機器間のデータ転送に用いられる規格。ドイツ　OSCH社が車載ネットワークとして提唱し、ISO11898およびISO11519として標準化された。車載ットワークの標準となっている。自動車以外の送用用機器や工業機器でも採用されている。
catkin	[ROS]独自のルドシステム
CUDA	*Compute Unified Device Architecture* NVIDIA社が提供する、GPU算を使った汎用計プラットフォームとプログラミングモデル。
DMI	*Distance Measuring Instrument* 走行離計。距行
DPM	*Deformable Part M dels* 物体検出手法
FlyCapture SDK	PointGrey社のカメラを制御するためのSDK。
FOT	*Field Operational Tests* 実際の運転環境下で、交通環境・ドライバ操作・車両動作の３者を長期間複数の車両運転者に跨って観測することで、運転を支援する技術や知識の有効を統計的に検証すること。新しい自動車技術開発方法論あり、全や環境など社会と深くかかわる自動車技術高高度要求に要である。特

用語	解説
GNSS	*Global Navigation Satellite System* 衛星測位システム。
IMU	*Inertial Measurement Unit* 慣性計測装置。角速度や加速度を計測する装置。
KF	*Kalman Filter* カルマンフィルター 誤差のある離散的な観測情報と、刻々と時間変化する量から目標物の位置を推定する手法
LIDAR	*Laser Imaging Detection and Ranging/レーザスキャナ/レーザレーダ* *レーザーをパルス状に発光し、その照射に対する散乱光を測定、遠距離(数100m程度)にある物体への距離を計測したり、その物体の立体形状(3Dイメージング(形状)や特性を測定することができる機器。*
Message	[ROS]ノード同士が通信する際のデータ構造。
NDT	*Normal Distributions Transform* 位置推定手法
Node	[ROS]単一の機能を提供するプロセス
Oculas	
Odometry	車輪の回転角や回転角速度などを用いて位置を推定する手法。
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
Point Cloud	3次元空間の点の集合（点群）データ。直交座標（x, y, z）で表現されている。物体の立体的な形状をこの点群を用いて表現できる。
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。
Quick Start	[Autoware]Autowareを起動した時に最初に表示されるGUI の最初のタブ。
ROS	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。
ROS PC	ROSとAutowareをインストールしたパソコン。
rosbag	[ROS]データロギングツール。拡張子は bag
rqt	[ROS]QtベースのGUIソフト開発ツール
Runtime Manager	[Autoware]Autowareのデベロッパ用ユーザインタフェース。Autoware起動時に表示される。
rviz	[ROS]データとソフトウェア状態の視覚化ツール。
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
TF	[ROS]座標変換ライブラリ。

用語	解説
Topic	[ROS]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。
way point	経路に定された1　m間隔の目印
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
高精度地図	MMS（Mobile Mapping System）を利用て得したデータを用いて生成した高精度な3　次元デジタル地図データ。従来の2次元データによる線表現に対し、曲線を表現できる3次元データであるとともに、5センチ以内という高い精度を持つ。
センサ・フュージョン	複数の能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
ベクタマップ	ベクターで道路などの地形情報を表現したＧＩ　Ｓ（Global Information System）データ。道路などの地形情報はジオメトリで扱われ、このジオメトリのベクターデータで構成されている。
メッセージパッシング	1つ以上の受信者に対して送信者がデータを配信するプロセス間通信の方式

チャプター
6

6. 関連文書

Autoware
http://www.pdsl.jp/fot/autoware/

AutowareRider

✧ 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

CarLink_CANusbAccessory_LS.apk
https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink_CANusbAccessory_LS.apk

CUDA
http://www.nvidia.com/object/cuda_home_new.html
http://www.nvidia.co.jp/object/cuda-jp.html
http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/
http://developer.nvidia.com/cuda-downloads

FlyCapture SDK
http://www.ptgrey.com/flycapture-sdk

OpenCV
http://opencv.org/
http://opencv.jp/
http://sourceforge.net/projects/opencvlibrary/

Qt
http://www.qt.io/
http://qt-users.jp/

ROS
http://www.ros.org/

Ubuntu Japanese Team
https://www.ubuntulinux.jp/

Ubuntu
http://www.ubuntu.com/

Velodyne ドライバ
https://github.com/ros-drivers/velodyne

デモ関連

- デモ用の launch ファイルを生成するスクリプト

  http://db3.ertl.jp/autoware/sample_data/my_launch.sh
- デモで使うデータ(守山地区の地図・キャリブレーション・経路)

  http://db3.ertl.jp/autoware/sample_data/sample_moriyama_data.tar.gz
- ROSBAG データ

  http://db3.ertl.jp/autoware/sample_data/sample_moriyama_150324.tar.gz
  注) この ROSBAG データには画像情報が含まれていないため、物体検出(Detection)はできません