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

1. このドキュメントについて 4

2. ROSとAutoware 5

ロボット用のミドルウエア ROS 5

ROSの特徴 6

Autoware 7

3次元地図の作成と共有 8

自己位置推定（NDT：Normal Distributions Transform） 8

物体検出 9

経路生成 9

自律走行 9

ユーザインタフェース 9

ROSとAutoware 10

Autoware の構造 12

認知・判断 12

判断・操作・位置推定 13

経路探索 14

3次元地図などのデータの読み込み(DB、ファイル) 15

ドライバおよびセンサフュージョン 16

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 17

その他 17

3. ノードの作成 18

開発の流れ 18

パッケージを作成する 18

ノードを作成する 21

ノードをビルドする 23

ノードの動作確認をする 24

Runtime Manager 26

Computingタブから起動・終了するROSノードの追加例 26

ComputingタブからROSノードへ与えるパラメータの設定例 29

パラメータ追加例 34

小数値のパラメータおよびスライダー表示 38

パラメータをコマンドライン引数として出力する場合 43

パラメータ設定のその他のkind行指定 48

Quick Startタブのボタンで起動・終了するコマンドの設定例 50

Sensingタブのボタンで起動・終了するコマンドの設定例 55

4. 環境構築 58

インストール 58

OS 58

ROS 58

Velodyneドライバ 60

OpenCV 60

Qt（必要な場合） 61

CUDA（必要な場合） 62

FlyCapture（必要な場合） 63

Autoware 64

AutowareRider 65

canlib 66

SSHの公開鍵の作成 66

5. 用語 67

6. 関連文書 70

チャプター

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

# 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]](#footnote-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。

### 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）

1. 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。

1. 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

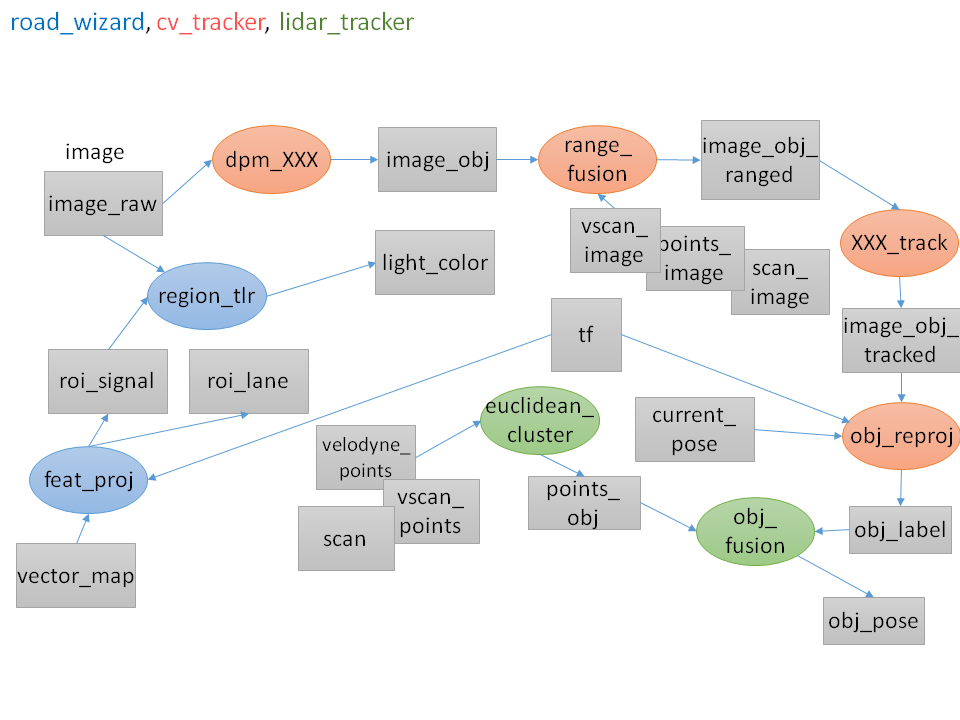


図 4 認知・判断

### 判断・操作・位置推定

ros/src/computing/perception/localization

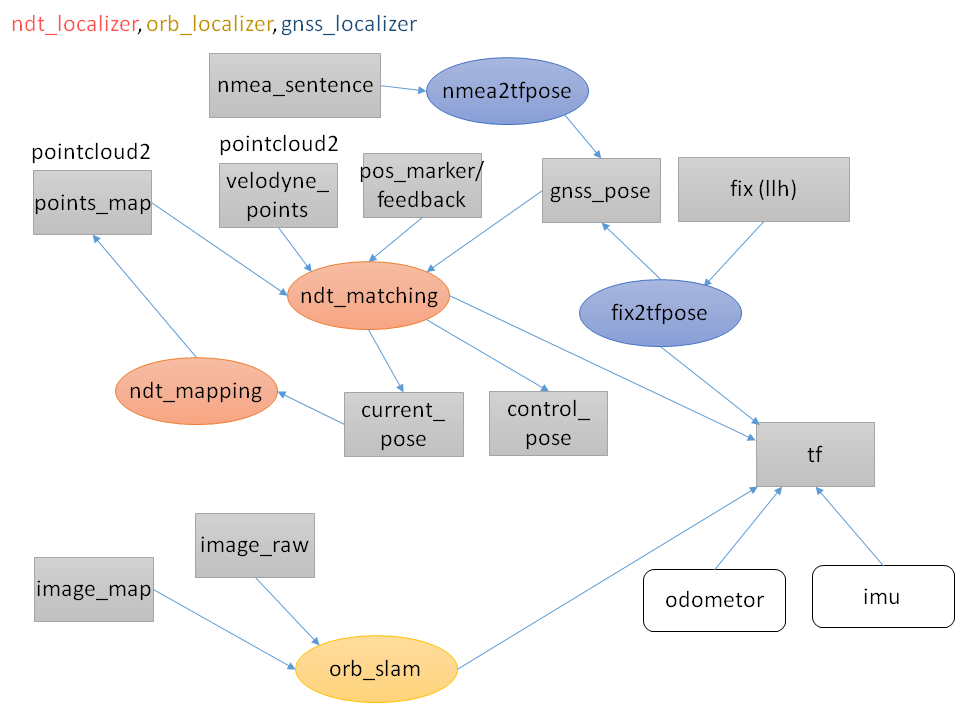


図 5　判断・操舵・位置推定

### 経路探索

ros/src/computing/planning

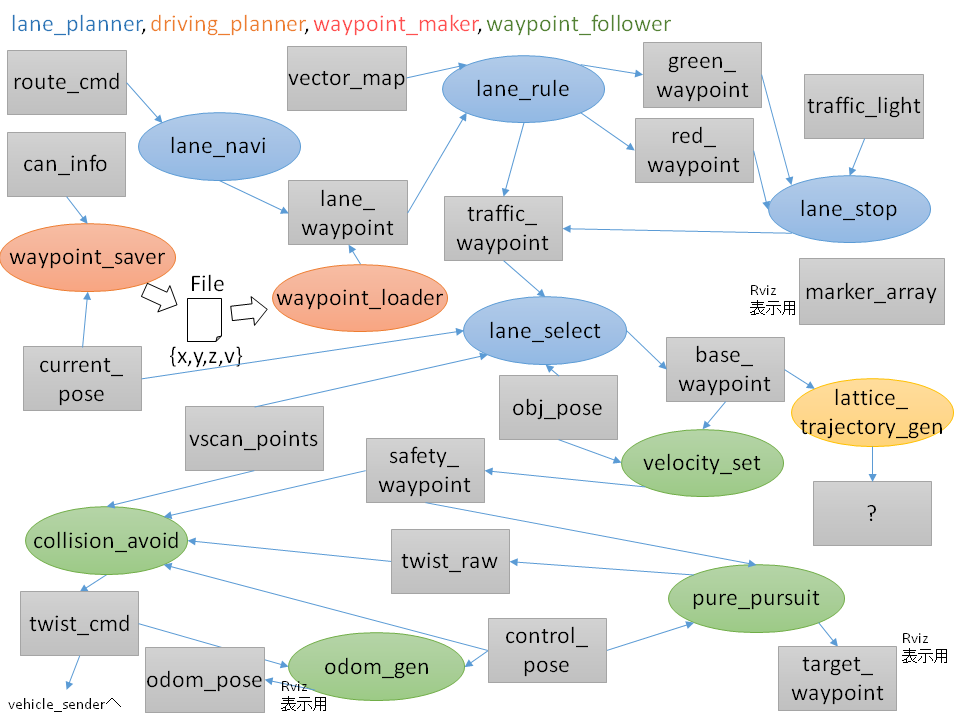


図 6 経路探索

### 3次元地図などのデータの読み込み(DB、ファイル)

ros/src/data

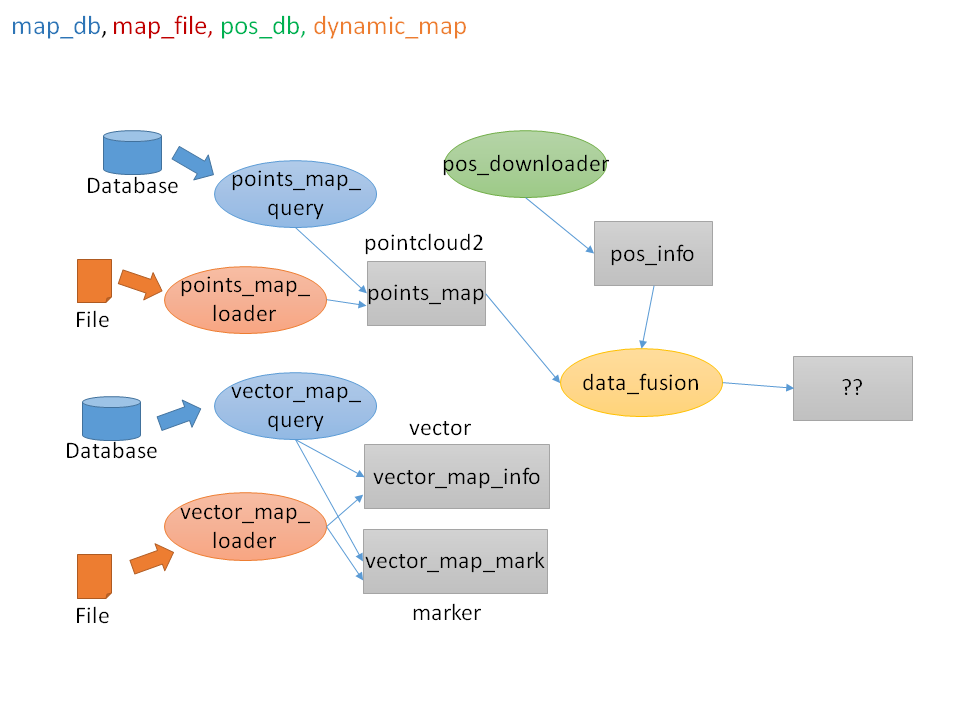


図 7 3次元地図などのデータの読み込み

### 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

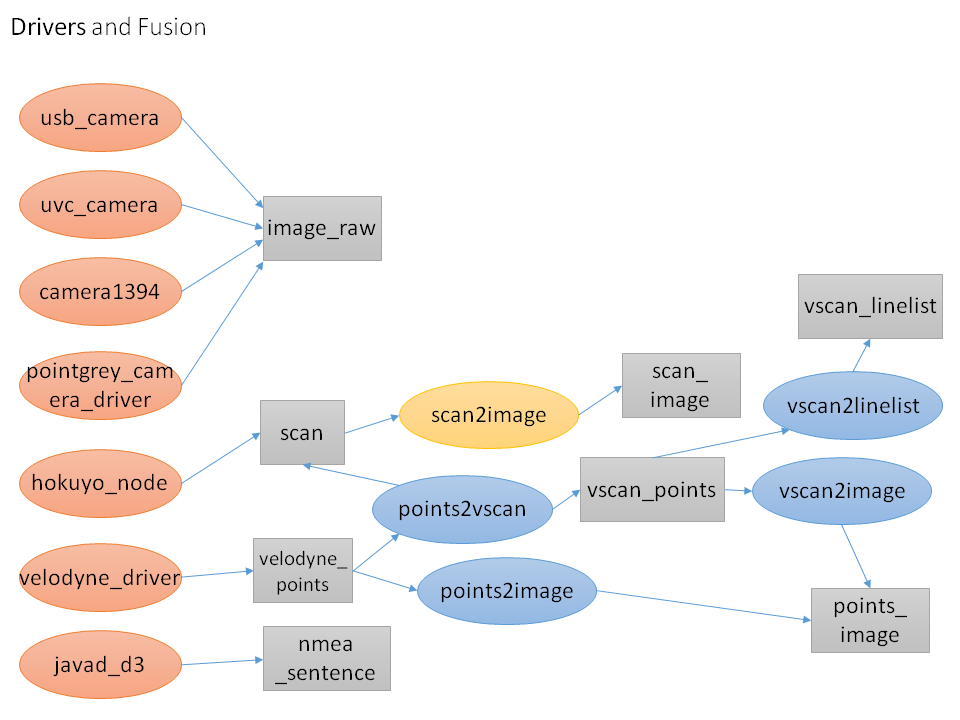


図 8 ドライバおよびセンサフュージョン

### 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

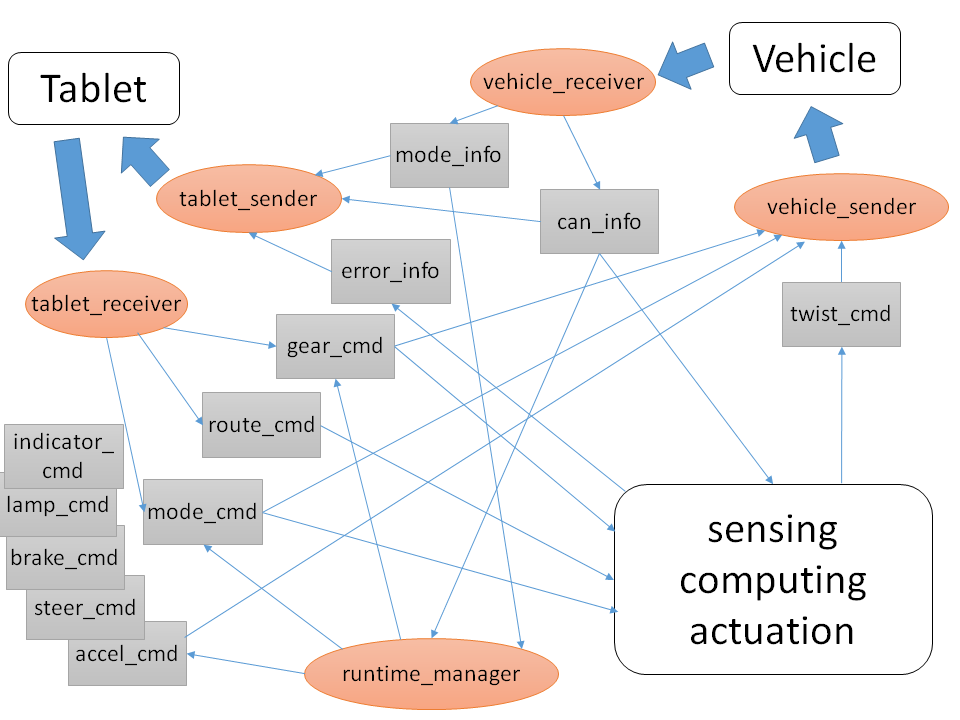


図 9 スマートフォン用アプリケーションとのインタフェース

### 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

1. 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。

1. Packageを作成する
2. ノードを作成する
3. ノードをビルドする
4. 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

1. 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。

1. 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

1. 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}")

1. 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)

1. 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}

)

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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

$

1. 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

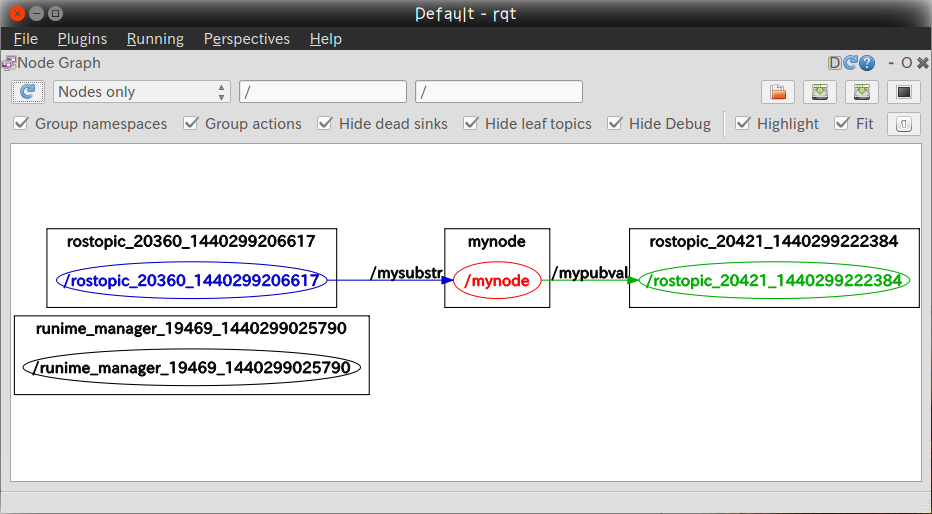
data: 11

---

1. 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

1. 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

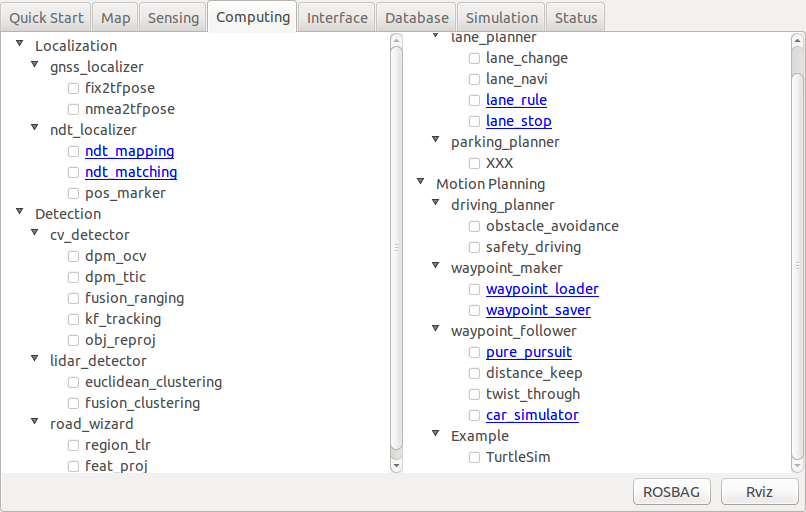


図 10 Comptuing タブ追加項目の表示

### 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。

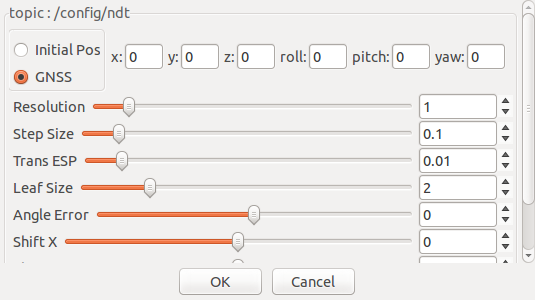


図 11 パラメータを調整するダイアログ

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

1. 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

1. 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。

1. 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>

:

1. 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。

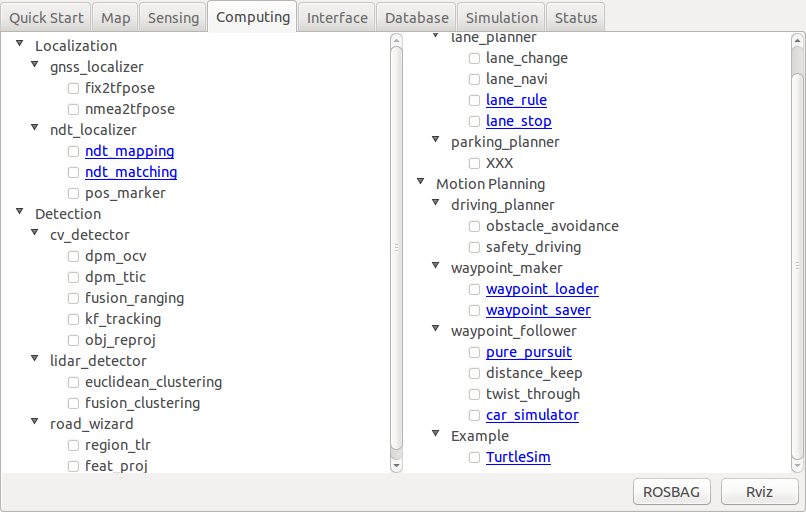


図 12 Computingタブ追加項目のリンク設定表示

1. 項目をクリックするとダイアログが表示されます。

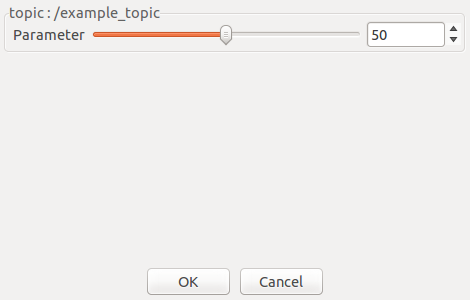


図 13 追加項目のパラメータ設定ダイアログ

1. 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

1. 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

1. 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。
2. kind行は、ファイルパス文字列を表す"path"を指定します。
3. v行は、デフォルト値のパスを指定します。
4. rosparam行は、rosparamパラメータの名前を指定します。
5. 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。

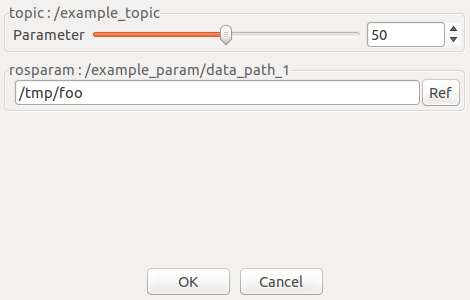


図 14 ファイルパス設定を追加したパラメータ設定ダイアログ

1. 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。

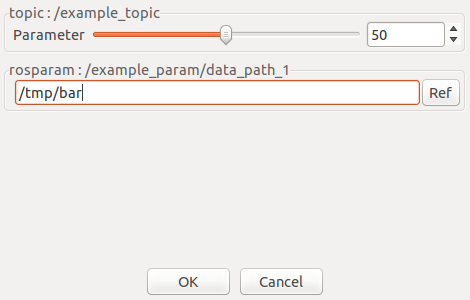


図 15 パラメータ設定ダイアログでパスを入力

1. 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

$

1. 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。
2. 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

1. 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

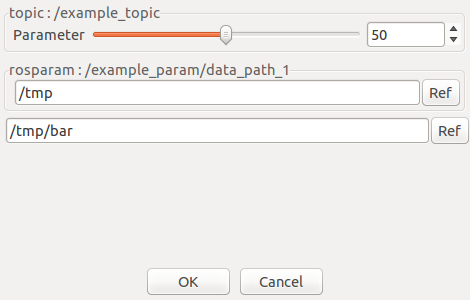


図 16 パラメータ設定ダイアログ

1. 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

1. 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

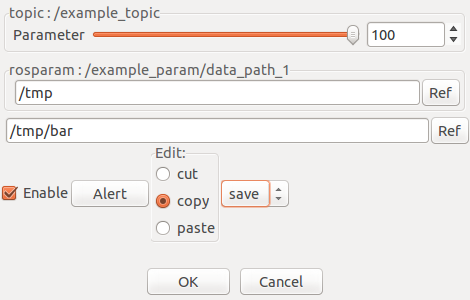


図 17 部品が追加されたパラメータ設定ダイアログ

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

1. 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" > \ /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-nmea-msgs \ ros-indigo-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

1. 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）

1. 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

1. 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）

1. 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

1. 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. 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

1. 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

1. インストーラを起動します。

$ cd flycapture2-2.6.3.4-amd64/

$ sudo sh install\_flycapture.sh

1. 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。

1. 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

1. 用語







チャプター

6

1. 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

1. <https://en.wikipedia.org/wiki/Willow_Garage> [↑](#footnote-ref-1)