Autoware デベロッパーズマニュアル

2016/APR/11

名古屋大学

内容

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」のデベロッパーズマニュアルです。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

- 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。
- Ladybug SDK
  PointGrey 社のカメラ「Ladybug」を制御するための SDK。
- 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。
- KLT (Kanade–Lucas–Tomasi feature tracker)
  特徴点を抽出し追跡を行う手法。
- 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

- Autoware
  http://www.pdsl.jp/fot/autoware/
- ROS
  http://www.ros.org/
- OpenCV
  http://opencv.org/
  http://opencv.jp/

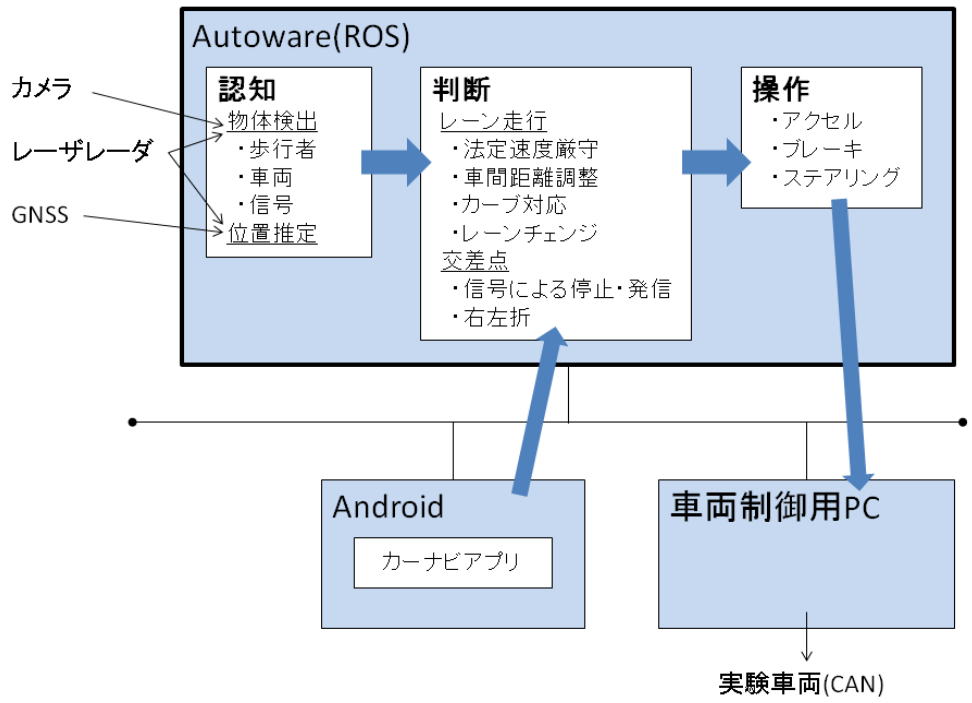
- Qt
  http://www.qt.io/
  http://qt-users.jp/
- CUDA
  http://www.nvidia.com/object/cuda_home_new.html
  http://www.nvidia.co.jp/object/cuda-jp.html
- FlyCapture SDK
  https://www.ptgrey.com/flycapture-sdk
- Ladybug SDK
  https://www.ptgrey.com/ladybug-sdk

問い合わせ先

Autoware Developers (autoware@googlegroups.com)

全体構成

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・判断
- ros/src/computing/planning/
  判断・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 や KLT 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、FlyCapture SDK および canlib 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 14.04
- Ubuntu 15.04

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-nmea-navsat-driver ros-indigo-sound-play
$ sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev \
    libssh2-1-dev libarmadillo-dev libpcap-dev gksu
```

- Ubuntu15.04 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-key adv --keyserver hkp://pool.sks-keyservers.net:80 \
    --recv-key 0xB01FA116
$ sudo apt-get install ros-jade-desktop-full ros-jade-nmea-msgs \
    ros-jade-nmea-navsat-driver ros-jade-sound-play
```

```
$ sudo apt-get install libnlopt-dev freeglut3-dev qt5-default libqt5opengl5-dev \
    libssh2-1-dev libarmadillo-dev libpcap-dev gksu
```

- ~/.bashrc などに以下を追加します。

  Ubuntu14.04 の場合:

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

  Ubuntu15.04 の場合:

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

CUDA

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”

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。

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。

- 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

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(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. 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。

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大まかな手順を示します。
基本的には、ROS のノード生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

開発の流れは以下の通りです。

1. パッケージを作成する
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行う

パッケージの作成

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 つのパッケージのみ存在する (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 や ノード名 1.py などのソースコード

ノード名 2/

ノード名 2.cpp や ノード名 2.py などのソースコード

...

```
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

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。

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
```

パッケージの依存関係は、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
…
```

ノードの作成

ここでは、簡単なノード 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 "ros/ros.h"
#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)のトピックを購読し、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。

ビルド

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}")
```

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)
```

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}
)
```

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
```

動作確認

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
```

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
```

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
```

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
```

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
```

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
$
```

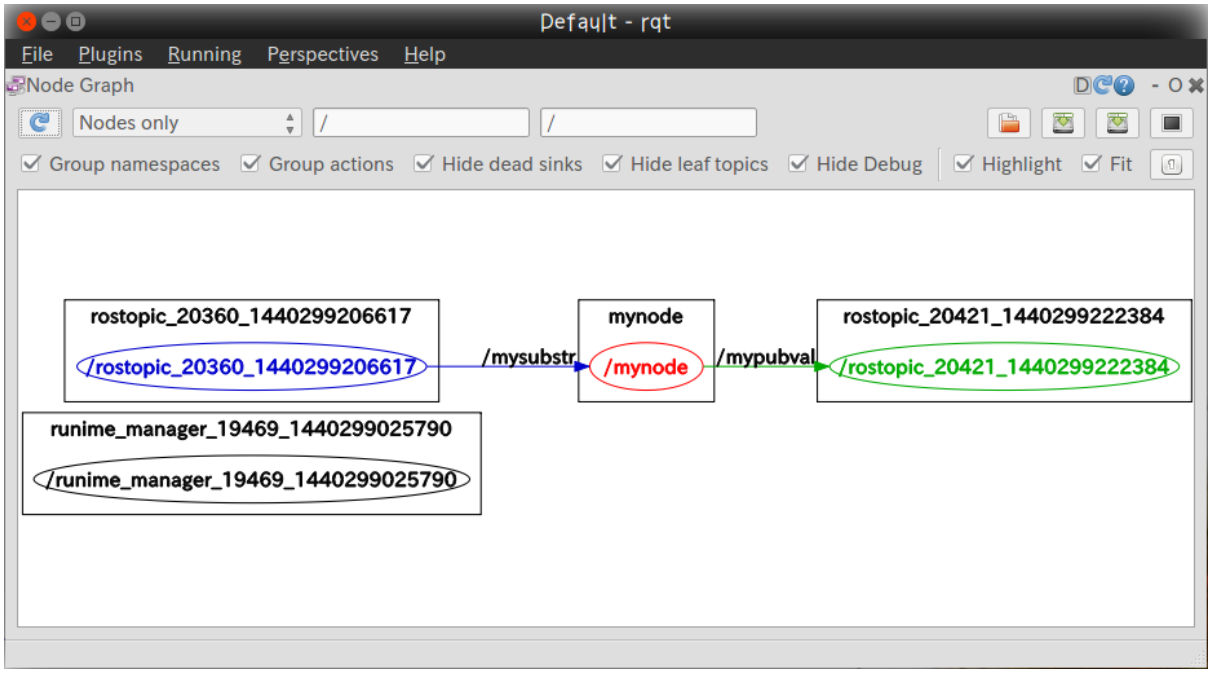
mynode が”hello world”を購読し、文字数を mypubval トピックに配信します。よって、先ほどの rostopic echo が以下を出力します。

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

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

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

概要

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す。

追加・変更例

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る。

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

例えば、Localization/ndt_localizer 欄 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る。

```
name : Computing
subs :
  :
〈略〉
  :
      - name : ndt_localizer
        subs :
  :
〈略〉
  :
      - name : ndt_matching
            cmd  : roslaunch ndt_localizer ndt_matching.launch
            param: ndt
```

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 スクリプトを起動する。

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る。

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・終了させる場合について、設定の追加例を示す。

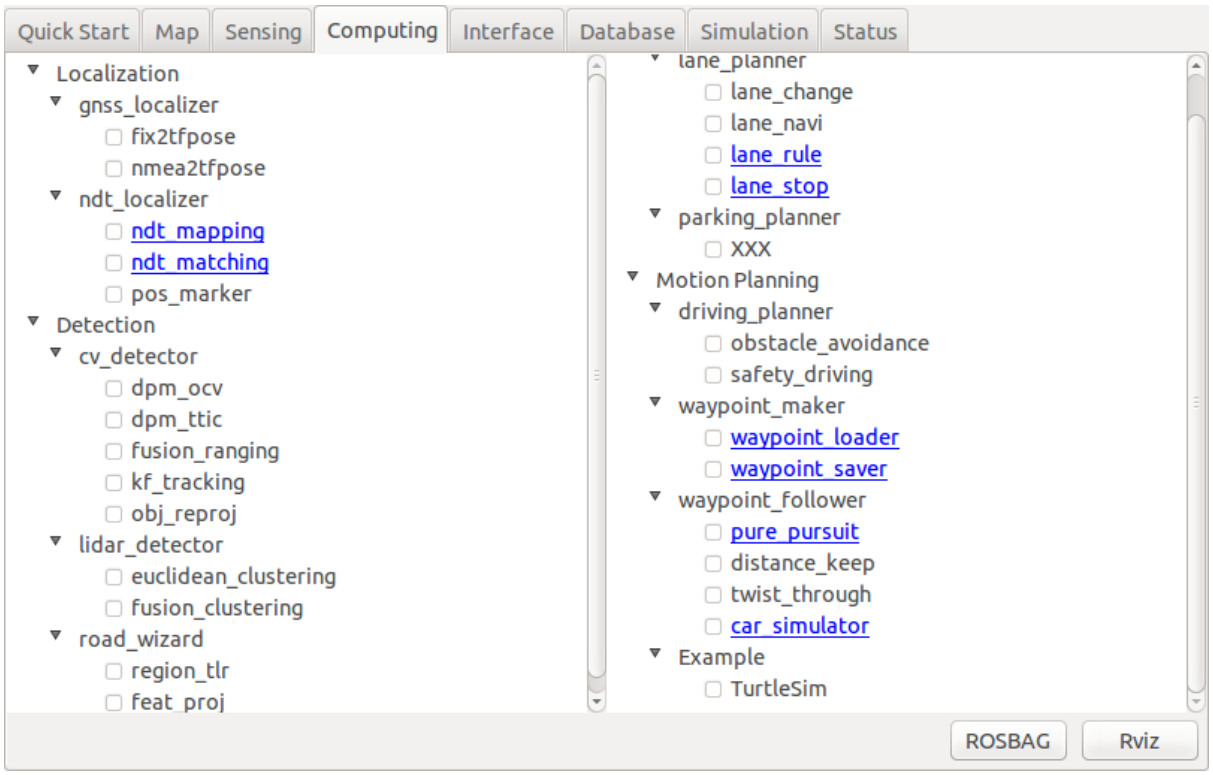
```
name : Computing
subs :
  :
〈略〉
  :
  - name : Motion Planning
    subs :
      - name : driver_lanner
        subs :
        - name : obstacle_avoidance
          cmd  : rosrun driving_lanner obstacle_avlidance
  :
〈略〉
  :
        - name : car_simulation
          cmd  : roslaunch waypoint_follower 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

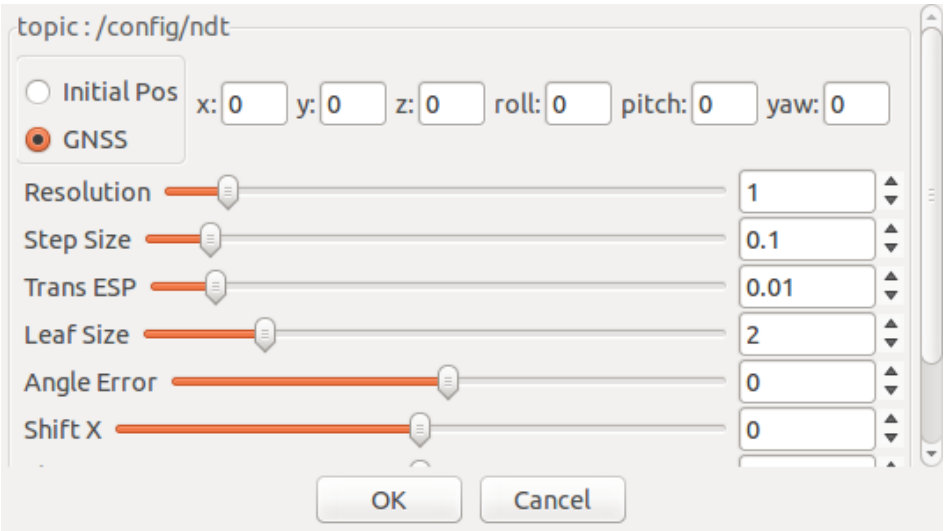
cmd : rosrun turtlesim turtlesim_node # この行を追加



Computing タブ追加項目の表示

Computing タブから起動する 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る。



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る。

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る。

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

Localization/ndt_localizer 欄 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る。

```
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る。

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る。

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す。

まず、設定ファイルに TrutleSim 項目を追加する。

```
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
```

次に、パラメータ名 example_param を指定する param 行を追加する。

```
        - name : Example
          subs :
          - name : TurtleSim
            cmd  : rosrun turtlesim turtlesim_node
```

```
        param: example_param                # この行を追加
```

さらに、後方の params 行以降に、example_param の詳細設定を追加する。

```
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
```

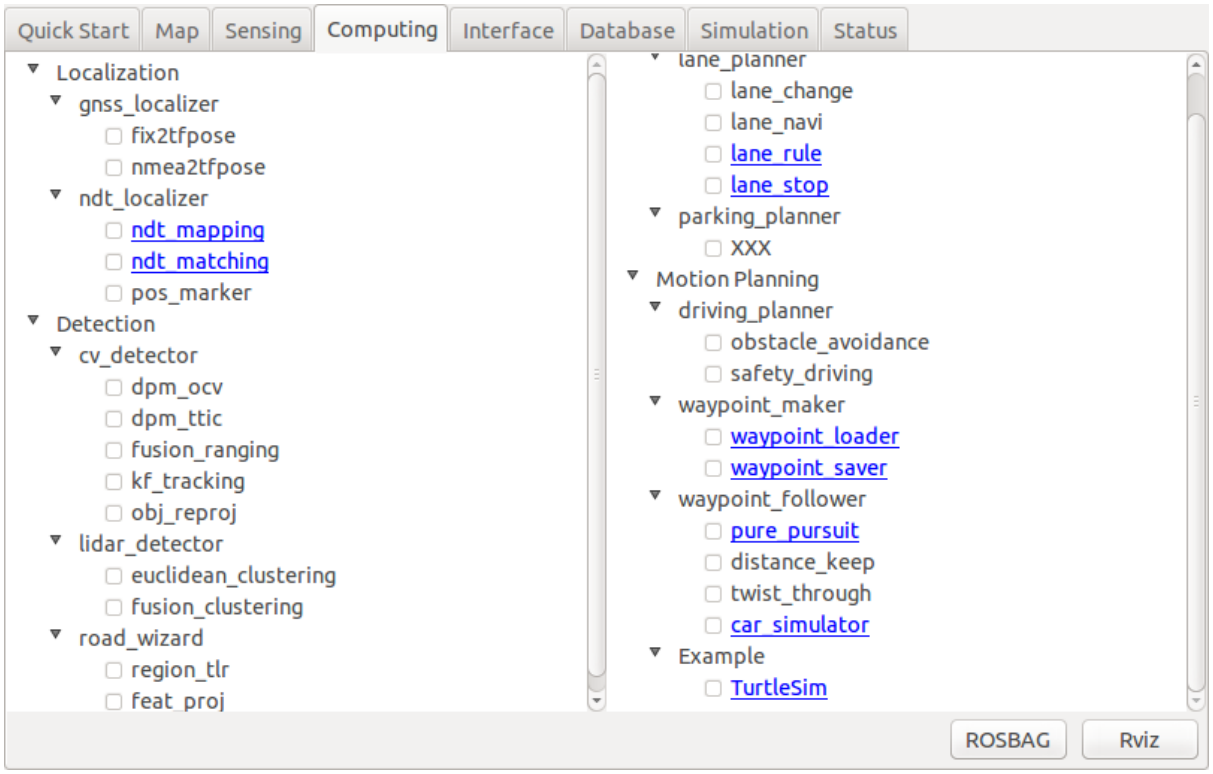
この例では、トピック名を /example、メッセージ型を Int32、メッセージ型 Int32 に含まれるメンバ data について、ダイアログに表示するラベル文字列を 'Parameter'、最小値を 0、最大値を 100、初期値を 50 に設定している。

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)
冒頭の include 行の箇所に、メッセージ型 Int32 の include 行を追加する。

```
  :
〈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
```

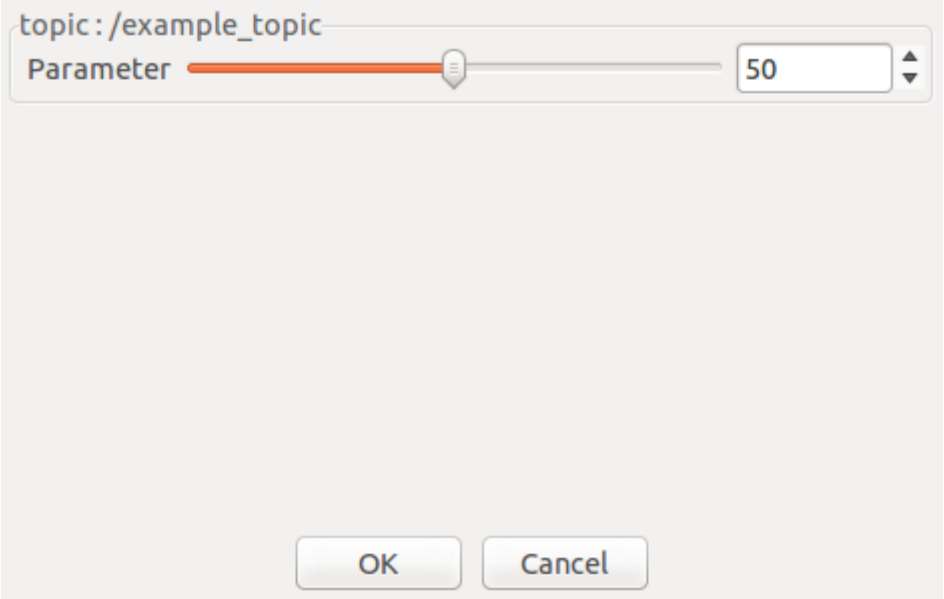
```
class MyFrame(rtmgr.MyFrame):
  :
〈略〉
  :
```

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る。



Computing タブ追加項目のリンク設定表示

項目をクリックするとダイアログが表示される。



追加項目のパラメータ設定ダイアログ

トピックを表示するため、別端末で次のコマンドを実行する。

```
$ rostopic echo /example_topic
```

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る。

```
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る。

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ない。

ダイアログで設定したパラメータを rosparam パラメータとして設定する例

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す。

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する。

設定ファイル computing.yaml にファイルパスの設定を追加する。

```
 :
〈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
```

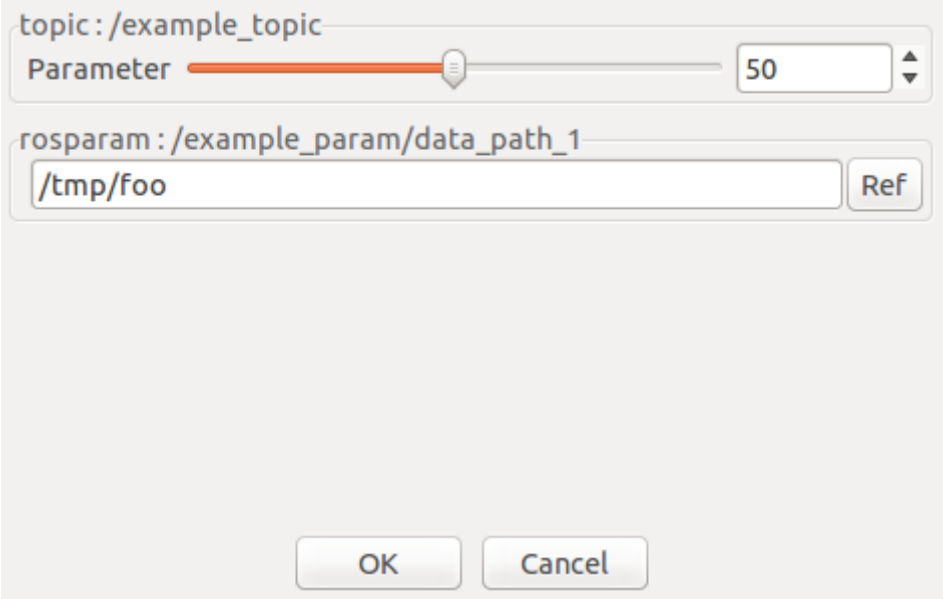
name 行は、トピックとしてメッセージ出力する場合は、メッセージ中のメンバ名を指定する。

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。

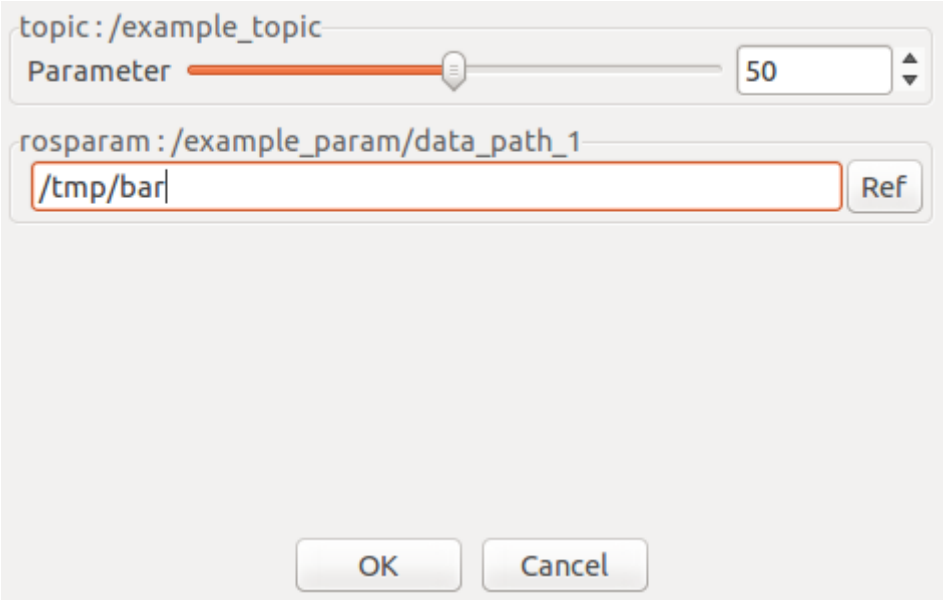
kind 行は、ファイルパス文字列を表す"path"を指定する。
v 行は、デフォルト値のパスを指定する。
rosparam 行は、rosparam パラメータの名前を指定する。

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る。



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

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る。



パラメータ設定ダイアログでパスを入力

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する。

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

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る。

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る。

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 指定を追加する。

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する。

```
$ 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る。

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る。

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す。

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"に変更しておく。

設定ファイル computing.yaml

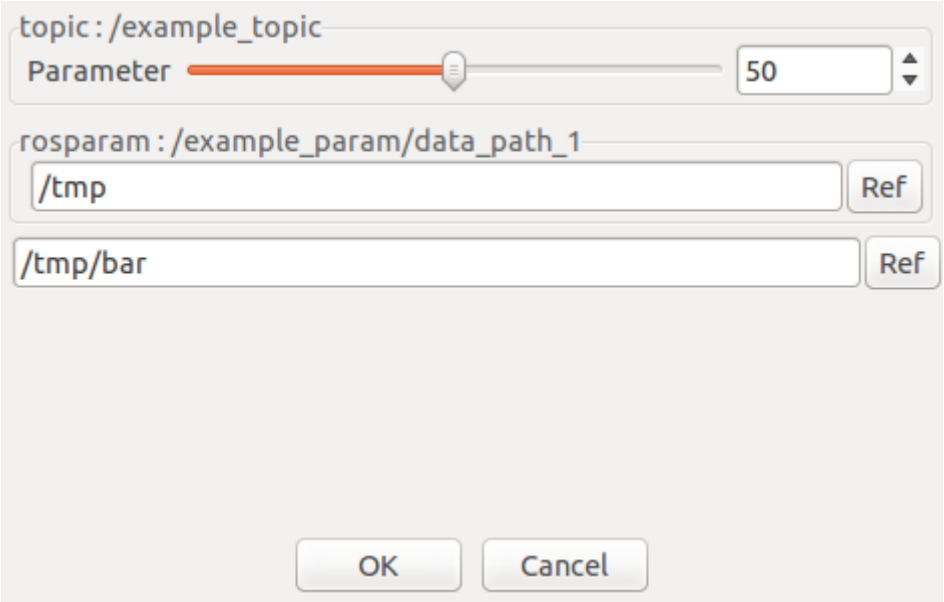
```
 :
〈略〉
 :
        - name : Example
          subs :
          - name : TurtleSim
          #cmd  : rosrun turtlesim turtlesim_node  # 変更
          cmd  : echo                              # 変更
            param: example_param
```

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する。

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
```



パラメータ設定ダイアログ

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、次の表示が出る。

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

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る。

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出来る。
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る。

<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> **の部分を出力しない。**
delim 行が存在しない場合は、<delim 行の値><パラメータの値> **の部分を出力しない。**
var_name 行が存在しない場合は、デフォルトとして name 行の値が使われる。

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> **だけになる。**

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> だけになる。**

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

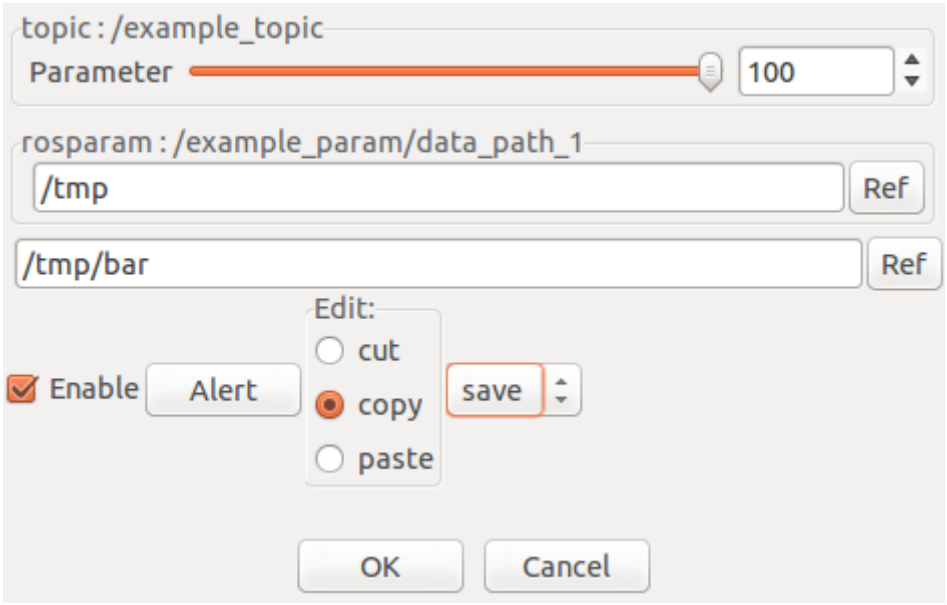
kind 行でチェックボックス、トグルボタン、ラジオボックス、メニューを指定出来る。

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)を扱う。

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
```



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ない。

追加したパラメータを出力するには、次の 3 つの方法がある。

- 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る。

```
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る。

```
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)を与えて実行する。

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)を終了させる。

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る。

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る。

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る。

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。

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る。

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る。

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る。

```
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する。

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 行を削除する。

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、次の表示が出る。

```
['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る。

```
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る。

```
  :
〈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"を実行する。

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)を終了させる。

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る。

```
  :
〈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、次の表示が出る。

```
['echo', 'hello']
hello
```

//