内容

[はじめに 2](#_Toc421132778)

[概要 2](#_Toc421132779)

[用語 2](#_Toc421132780)

[関連文書 3](#_Toc421132781)

[全体構成 4](#_Toc421132782)

[環境構築の手順 4](#_Toc421132783)

[Linux 4](#_Toc421132784)

[ROS 5](#_Toc421132785)

[OpenCV 5](#_Toc421132786)

[Qt 5](#_Toc421132787)

[CUDA 6](#_Toc421132788)

[FlyCapture2 7](#_Toc421132789)

[Autoware 8](#_Toc421132790)

[AutowareRider 8](#_Toc421132791)

[ノードの作成 10](#_Toc421132792)

[開発の流れ 10](#_Toc421132793)

[ノードの作成 10](#_Toc421132794)

[ビルド 10](#_Toc421132795)

[確認・デバッグ方法 10](#_Toc421132796)

[Runtime Manager 10](#_Toc421132797)

[概要 10](#_Toc421132798)

[追加・変更例 10](#_Toc421132799)

[Computingタブから起動・終了するROSノードの追加例 10](#_Toc421132800)

[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 12](#_Toc421132801)

共通

# 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

# 自動運転に関する用語も、統一したいので追加する。

* ROS (Robot Operating System)

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

* パッケージ (Package)

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

* ノード (Node)

単一の機能を提供するプロセス。

* メッセージ (Message)

ノード同士が通信する際のデータ構造。

* トピック (Topic)

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

* OpenCV (Open source Computer Vision library)

コンピュータビジョンを扱うための画像処理ライブラリ。

* Qt

アプリケーション・ユーザ・インタフェースのフレームワーク。

* CUDA (Compute Unified Device Architecture)

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

* FlyCapture SDK

PointGrey社のカメラを制御するためのSDK。

* FOT (Field Operation Test)

実道実験。

* GNSS (Global Navigation Satellite System)

衛星測位システム。

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

レーザー照射を利用して距離などを計測する装置。

* DPM (Deformable Part Model)

物体検出手法。

* KF (Kalman Filter)

過去の観測値をもとに将来の状態を推定する手法。

* NDT (Normal Distributions Transform)

位置推定手法。

* キャリブレーション

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

* センサ・フュージョン

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

* TF (TransForm?)

ROSの座標変換ライブラリ?

* オドメトリ(Odometry)

車輪の回転角と回転角速度を積算して位置を推定する手法。

* SLAM(Simultaneous Localization and Mapping)

自己位置推定と環境地図作成を同時に行うこと。

## 関連文書

# 文書ではなくURLになっていますが…

* Autoware

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

* ROS

<http://www.ros.org/>

* OpenCV

<http://opencv.org/>

<http://opencv.jp/>

* Qt

<http://www.qt.io/>

<http://qt-users.jp/>

* CUDA

<http://www.nvidia.com/object/cuda_home_new.html>

<http://www.nvidia.co.jp/object/cuda-jp.html>

* FlyCapture SDK

<http://www.ptgrey.com/flycapture-sdk>

# 全体構成

# AutowareのPC + 各種センサ機器 の図と説明を書く。

# Autowareの中は、加藤先生の仕様書を参考に。

# ただ、仕様書は膨大なので、機能をひとまとめにした方がいいかも。

# デモ内容とも絡みますが、こういう流れでこの機能が動くみたいな例を示す?

# 環境構築の手順

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

CUDAとFlyCapture SDKは、必須ではありません。

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

## Linux

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

* Ubuntu 13.04
* Ubuntu 13.10
* Ubuntu 14.04

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

* Ubuntu Japanese Team

<https://www.ubuntulinux.jp/>

* Ubuntu

<http://www.ubuntu.com/>

## ROS

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

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > \

/etc/apt/sources.list.d/ros-latest.list'

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install ros-indigo-desktop-full ros-indigo-velodyne-pointcloud \

ros-indigo-nmea-msgs

$ sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev

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

# sources.listの設定など必要

$ sudo apt-get install ros-hydro-desktop-full ros-hydro-velodyne-pointcloud \

ros-indigo-nmea-msgs

$ sudo apt-get install libnlopt-dev freeglut3-dev

1. ~/.bashrcなどに以下を追加します。

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

## OpenCV

OpenCVのサイト(<http://sourceforge.net/projects/opencvlibrary/>)からソースコードを入手し、以下の手順でインストールを行います。

# 現在、2.4.8が入手不可だが、他のバージョンでもOKか?

$ unzip opencv-2.4.8.zip

$ cd opencv-2.4.8

$ cmake .

$ make

$ sudo make install

## Qt

1. まず、Qt5に必要なパッケージを、以下の手順でインストールします。

$ sudo apt-get build-dep qt5-default

$ sudo apt-get install build-essential perl python git

$ sudo apt-get install "^libxcb.\*" libx11-xcb-dev libglu1-mesa-dev \

libxrender-dev libxi-dev

$ sudo apt-get install flex bison gperf libicu-dev libxslt-dev ruby

$ sudo apt-get install libssl-dev libxcursor-dev libxcomposite-dev libxdamage-dev \

libxrandr-dev libfontconfig1-dev

$ sudo apt-get install libasound2-dev libgstreamer0.10-dev \

libgstreamer-plugins-base0.10-dev

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 https://git.gitorious.org/qt/qt5.git qt5

$ cd qt5/

$ git checkout v5.2.1

$ perl init-repository --no-webkit

(webkit は大きいため、--no-webkitを指定しています)

$ ./configure -developer-build -opensource -nomake examples -nomake tests

(ライセンスを受諾する必要があります)

$ make -j

(ビルドには数時間かかります)

$ make install

$ sudo cp -r qtbase /usr/local/qtbase5

## CUDA

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

1. 環境の確認

$ lspci | grep -i nvidia

(NVIDIAのボードの情報が出力されることを確認)

$ uname -m

(x86\_64であることを確認)

$ gcc --version

(インストールされていることを確認)

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”

## FlyCapture2

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

# 2014年10月28日に試したときの手順

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

1. PointGrey社のサイト(<http://www.ptgrey.com/>)から、FlyCapture SDKをダウンロードします。(ユーザ登録が必要です。)
2. 以下の手順で、事前にパッケージをインストールします。

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

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

This is a script to assist with installation of the FlyCapture2 SDK.  
 Would you like to continue and install all the FlyCapture2 SDK packages?  
 (y/n)$ y ← 「y」と答えます  
 ...  
 Preparing to unpack updatorgui-2.6.3.4\_amd64.deb ...  
 Unpacking updatorgui (2.6.3.4) ...  
 updatorgui (2.6.3.4) を設定しています ...  
 Processing triggers for man-db (2.6.7.1-1ubuntu1) ...  
 Would you like to add a udev entry to allow access to IEEE-1394 and USB hardware?  
 If this is not ran then your cameras may be only accessible by running flycap as sudo.  
 (y/n)$ y ← 「y」と答えます

## Autoware

以下の手順でAutowareを入手し、ビルドおよびインストールを行います。

$ git clone https://github.com/CPFL/Autoware.git  
 $ cd Autoware/ros/src  
 $ catkin\_init\_workspace  
 $ cd ../  
 $ ./catkin\_make\_release

## AutowareRider

以下のURLからAPKファイルを入手し、インストールを行います。

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

デベロッパーズマニュアル

# ノードの作成

## 開発の流れ

# パッケージ作成の決め事なども

## ノードの作成

## ビルド

## 確認・デバッグ方法

# rviz や rosgraph, rostopic, rosnode,...

# 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\_launch\_cmd.yaml

例えば、Perception/Detection欄car\_dpm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 : Perception

subs :

- name : Detection

subs :

- name : car\_dpm

cmd : rosrun car\_detector car\_dpm

param: car\_dpm

car\_dpm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"rosrun car\_detector car\_dpm"を実行し、car\_detectorパッケージのcar\_dpmノードを起動する。

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car\_dpmノードを終了させる。

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 : Planning

subs :

- name : Path

subs :

- name : lane\_navi

cmd : rosrun lane\_planner lane\_navi

param: lane\_navi

:

<略>

:

- name : waypoint\_loader

cmd : roslaunch waypoint\_maker waypoint\_loader.launch

param: waypoint\_loader

gui :

waypoint\_filename :

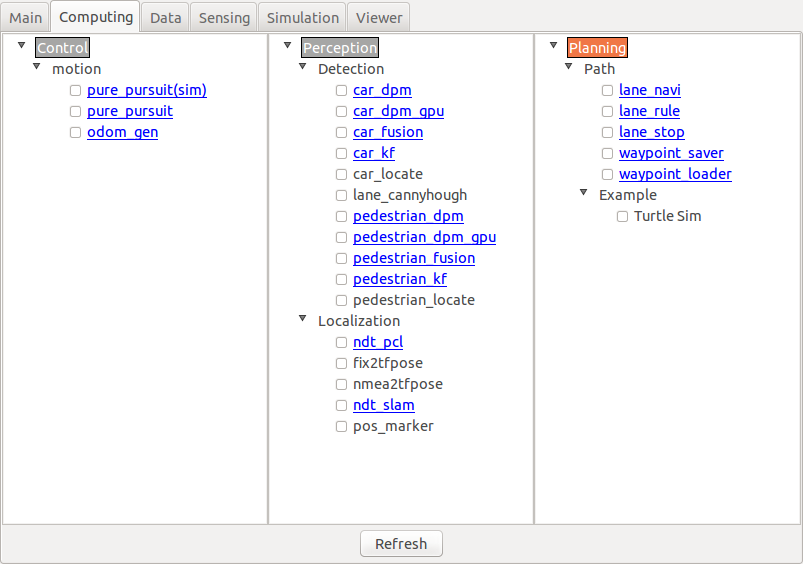
prop : 1

- 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

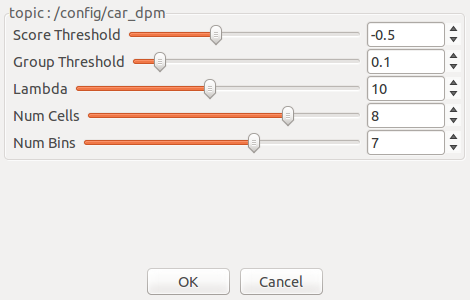


Computingタブ追加項目の表示

### 

### 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、Perception/Detection欄car\_dpm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/car\_dpm として発行され、起動しているcar\_dpm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\_launch\_cmd.yaml

Perception/Detection欄car\_dpm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 : Perception

subs :

- name : Detection

subs :

- name : car\_dpm

cmd : rosrun car\_detector car\_dpm

param: car\_dpm

param行の car\_dpm の記述は、パラメータ名が car\_dpm 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 : car\_dpm" に記述されている事を表す。

params :

- name : car\_dpm

topic : /config/car\_dpm

msg : ConfigCarDpm

vars :

- name : score\_threshold

label : Score Threshold

min : -2

max : 2

v : -0.5

- name : group\_threshold

label : Group Threshold

min : 0

max : 1

v : 0.1

- name : Lambda

label : Lambda

min : 1

max : 20

v : 10

- name : num\_cells

label : Num Cells

min : 2

max : 10

v : 8

- name : num\_bins

label : Num Bins

min : 2

max : 10

v : 9

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。

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 : Planning

subs :

- name : Path

subs :

- name : lane\_navi

cmd : rosrun lane\_planner lane\_navi

param: lane\_navi

:

<略>

:

- name : waypoint\_loader

cmd : roslaunch waypoint\_maker waypoint\_loader.launch

param: waypoint\_loader

gui :

waypoint\_filename :

prop : 1

- 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

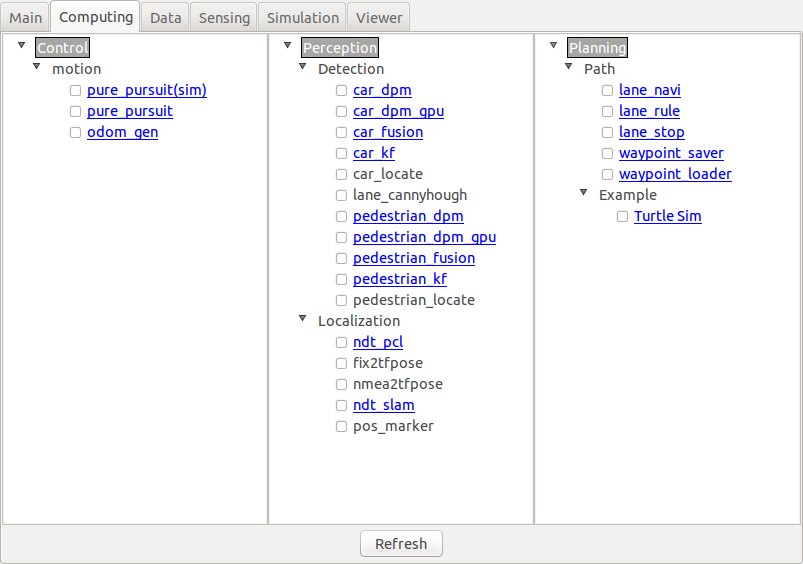
class MyFrame(rtmgr.MyFrame):

:

<略>

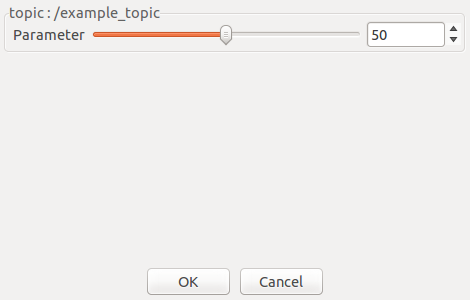
:

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

---