Autoware Documentation

Release 1.9.0

Autoware Community

Nov 07, 2018

Contents

Warning: Under Construction

Autoware is the world's first "all-in-one" open-source software for self-driving vehicles. The capabilities of Autoware are primarily well-suited for urban cities, but highways, freeways, mesomountaineous regions, and geofenced areas can be also covered. The code base of Autoware is protected by the BSD License. Please use it at your own discretion. For safe use, we provide a ROSBAG-based simulation environment for those who do not own real autonomous vehicles. If you plan to use Autoware with real autonomous vehicles, **please formulate safety measures and assessment of risk before field testing.**

CHAPTER 1

Users Guide

1.1 Installation

1.1.1 Spec Recommendation

- **Generic x86_64**
  - Intel Core i7 (preferred), Core i5, Atom
  - 16GB to 32GB of main memory
  - More than 30GB of SSD
  - NVIDIA GTX GeForce GPU (980M or higher performance)
- **NVIDIA DRIVE**
  - DRIVE PX2, DRIVE Xavier (on the way)
  - More than 30GB of SSD

1.1.2 Build

There are three choices to build Autoware. Using Autoware *Docker Image*, which provides built Autoware, is strongly recommended. In case that you do not want to or can not use Docker, please follow the instruction provided by *Source Build* or *Cross Build*.

Docker Image

Generic x86_64

Docker Setup

You first need to install Docker CE.
https://docs.docker.com/install/

NVIDIA Docker Setup

You need to install Docker Plugin provided by NVIDIA in order to access NVIDIA GPUs from Docker Container. If not already installed, Install CUDA drivers for your platform: Linux

1. Download the .deb file.

```
$ wget https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.1/nvidia-docker_
↪1.0.1-1_amd64.deb
```

2. Install the downloaded .deb file.

```
$ sudo dpkg -i nvidia-docker_1.0.1-1_amd64.deb
```

3. Check if the nvidia-docker service exists.

```
$ systemctl list-units --type=service | grep -i nvidia-docker
```

4. Check if the nvidia-docker service runs.

```
$ sudo nvidia-docker run --rm nvidia/cuda nvidia-smi
```

Note: If you run into the following error, try following these nvidia-docker instructions.

> docker: Error response from daemon: OCI runtime create failed: container_linux.go:348: starting container process caused "exec: "nvidia-smi": executable file not found in $PATH": unknown.

Autoware Docker Setup

Generic aarch64

Docker Setup

You first need to install Docker CE.
https://docs.docker.com/install/

Autoware Docker Setup

NVIDIA DRIVE

We provide Docker environments for the NVIDIA DRIVE platform with support from NVIDIA Corporation. To access any of the following environments, you should have signed their NDA/SLA to access NVIDIA DevZone and, in some case, should have been authorized to access internal details of NVIDIA DriveWorks SDK.

DRIVE PX2

If you are using the DRIVE PX2 platform, you can choose Docker with or without NVIDIA DriveWorks. With NVIDIA DriveWorks, you may leverage NVIDIA self-driving capabilities, such as line detection and object detection using DriveNet. Without DriveWorks, on the other hand, all self-driving capabilities are covered by Autoware, thus for example object detection is based on SSD or YOLO2. For more details, try it our yourself.

Note: DRIVE PX2 requires you to be a licensee of NVIDIA DRIVE and DevZone. To complete the installation process introduced below, please contact Autoware Developers Group at autoware@googlegroups.com.

Docker Setup

You first need to setup the Docker environment. You may not access docker.io with the default configuration of Ubuntu 16.04, so try the following installation process.

```
$ sudo apt-get install -y software-properties-common
$ sudo apt-add-repository universe
$ sudo apt-get update
$ sudo apt-get install docker.io
$ sudo apt-get update
```

Type the following commands.

```
$ sudo docker info | grep "Docker Root Dir"
```

If you get the following output, you might continue with the process.

```
Docker Root Dir: /var/lib/docker
```

Autoware Setup

Todo: Insert document

DRIVE Xavier

Coming soon.

Source Build

Requirements

- ROS Indigo (Ubuntu 14.04) or ROS Kinetic (Ubuntu 16.04)
- OpenCV 2.4.10 or higher
- Qt 5.2.1 or higher
- CUDA (optional)
- FlyCapture2 (optional)
- Armadillo (optional)

ROS

Todo: Insert link to ROS install

CUDA

Todo: Insert link to CUDA install

Install system dependencies for Ubuntu 14.04 Indigo

```
% sudo apt-get install -y  python-catkin-pkg python-rosdep python-wstool ros-$ROS_
↪DISTRO-catkin
% sudo add-apt-repository ppa:mosquitto-dev/mosquitto-ppa
% sudo apt-get update
% sudo apt-get install libmosquitto-dev
```

Note: NOTE: Please do not install ros-indigo-velodyne-pointcloud package. If it is already installed, please uninstall.

Install system dependencies for Ubuntu 16.04 Kinetic

```
% sudo apt-get update
% sudo apt-get install -y python-catkin-pkg python-rosdep python-wstool ros-$ROS_
↪DISTRO-catkin libmosquitto-dev
```

How to Build

1. Clone the repository

```
$ git clone https://github.com/CPFL/Autoware.git --recurse-submodules
```

Note: If you already have a copy of the repo, run `$ git submodule update --init --recursive`.

2. Initialize the workspace, let rosdep to install the missing dependencies and compile.

```
$ cd Autoware/ros/src
$ catkin_init_workspace
$ cd ../
$ rosdep install -y --from-paths src --ignore-src --rosdistro $ROS_DISTRO
$ ./catkin_make_release
```

Cross Build

1.1.3 Setup

Some nodes require additional setup. Please follow the instructions if you want to use the nodes.

Setup DNN-based nodes

DNN-based nodes require their pretrained models. The nodes in Autoware are:

- YOLO
- SSD

Todo: Delete link to GitHub and Import README.md in vision_darknet_detect and vision_ssd_detect.

Please follow README.md in each package to install the models if you need these nodes.

Warning: Autoware itself is licensed under BSD 3-Clause "New" or "Revised" License. However, pre-trained models of DNN that the users of Autoware use are expected to be licensed under another license. For example, KITTI is licensed under CC BY-NC-SA, which do not allow commercial uses. **Please follow each license of the models you use.**

1.2 Tutorials

1.2.1 How to Start

The first tutorial shows you how to run the user interface. GUI named Runtime Manager is launched by the following commands with two terminals.

```
$ cd Autoware/ros
$ ./run
```

If everything goes well, you can find the windows shown in the following figure.

Todo: Insert a figure.

1.2.2 Quick Start

Demo data

This demo will require 3D map and ROSBAG data. Please download the following sample demo data before running the demo.

1. Download the sample 3D pointcloud/vector map data. [link]
2. Download the sample ROSBAG data (LiDAR: VELODYNE HDL-32E, GNSS: JAVAD GPS RTK Delta 3). [link]

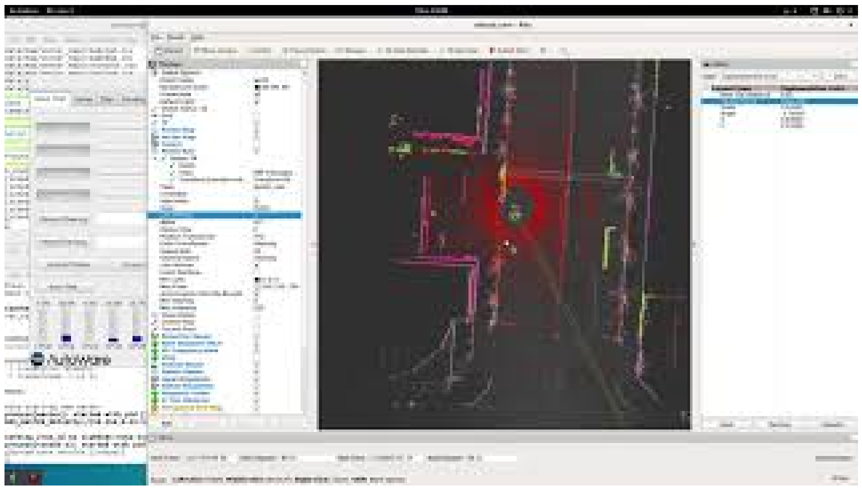
Note: Want more data?

Once the demo goes well, you can visit ROSBAG STORE to get more data. Please also consider your contribution to this data sharing service by uploading your ROSBAG data.

Demo run

Autoware provide a set of the preinstalled roslaunch scripts for the demo. Please follow the steps below:

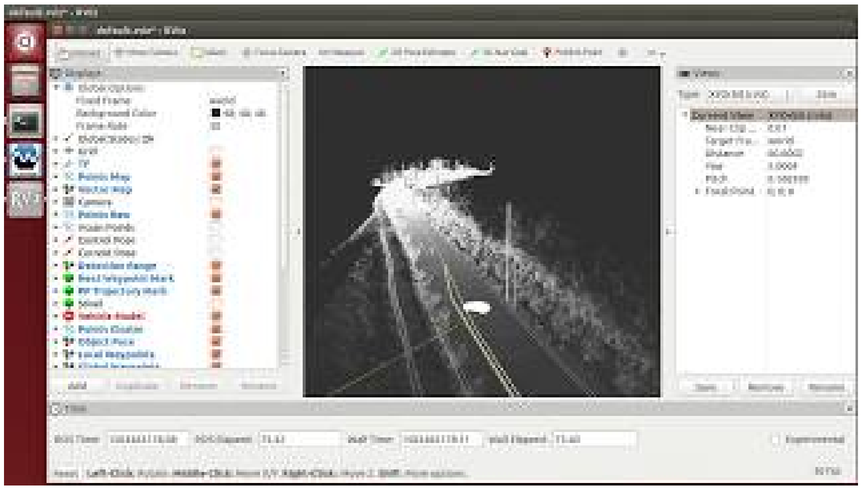
1. Go to the Simulation tab of Autoware Runtime Manager, and load the sample ROSBAG data.
2. Play the loaded ROSBAG data, and immediately pause it once.
3. Launch RViz.
4. Go to the Quick Start tab of Autoware Runtime Manager, and load the preinstalled roslaunch scripts one by one.

Please follow the instruction video below: 

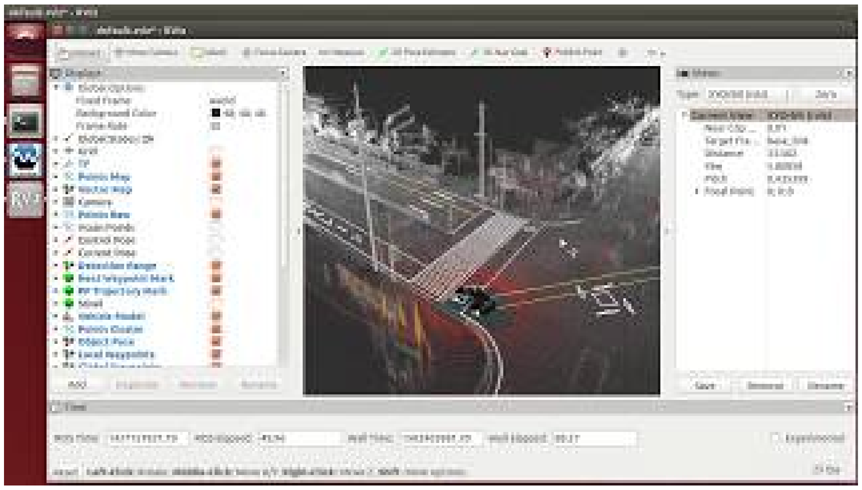
1.3 Videos

1.3.1 Quick Start

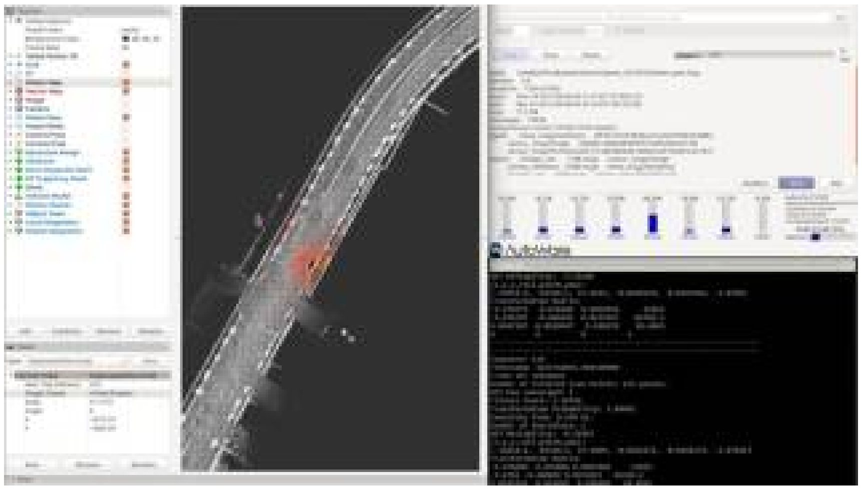
1.3.2 Loading Map Data



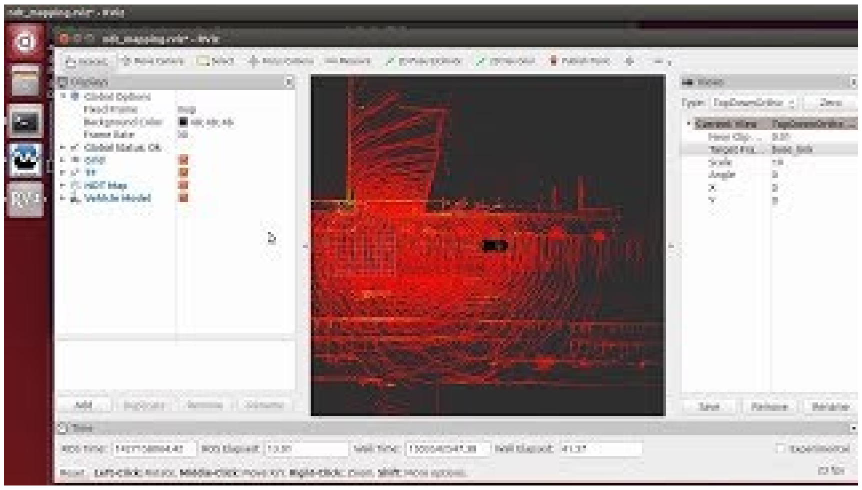
1.3.3 Localization with GNSS



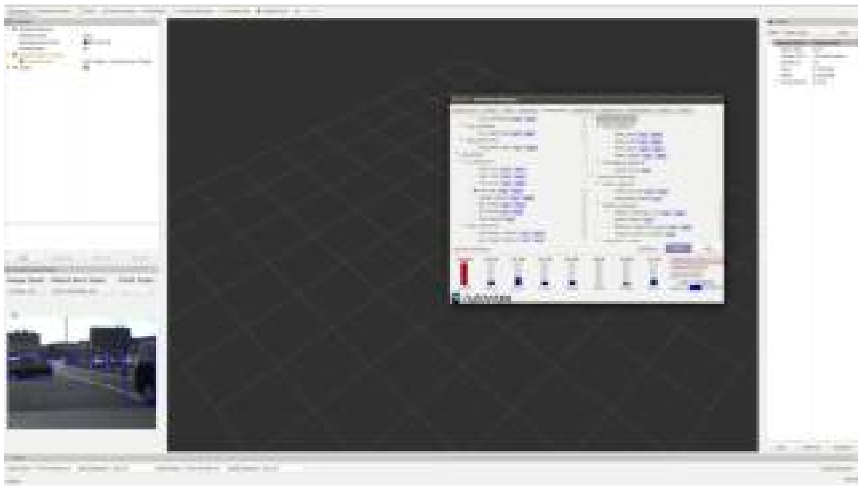
1.3.4 Localization without GNSS



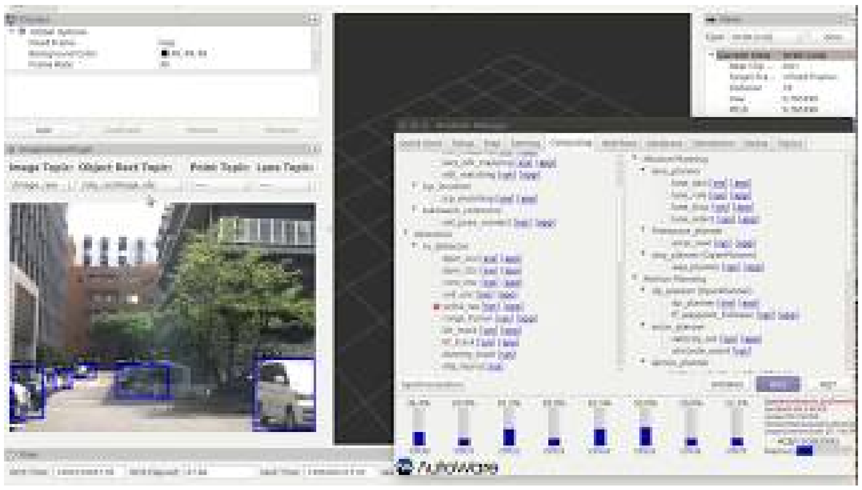
1.3.5 Mapping



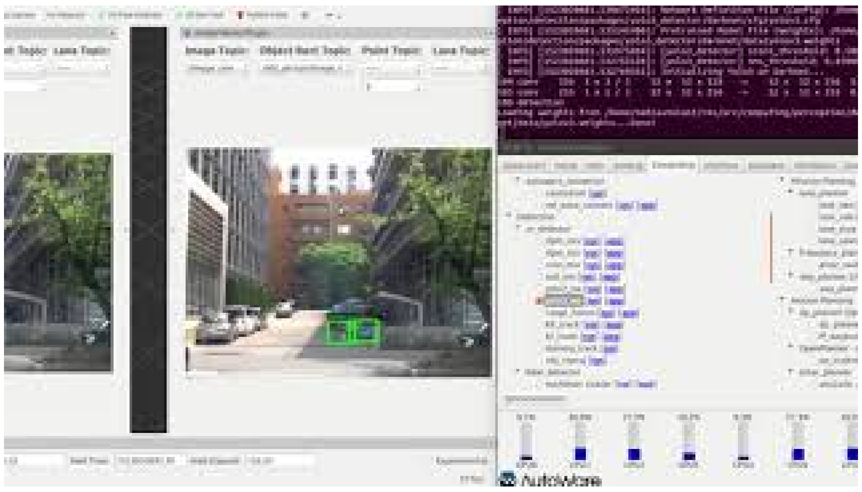
1.3.6 Detection with SSD



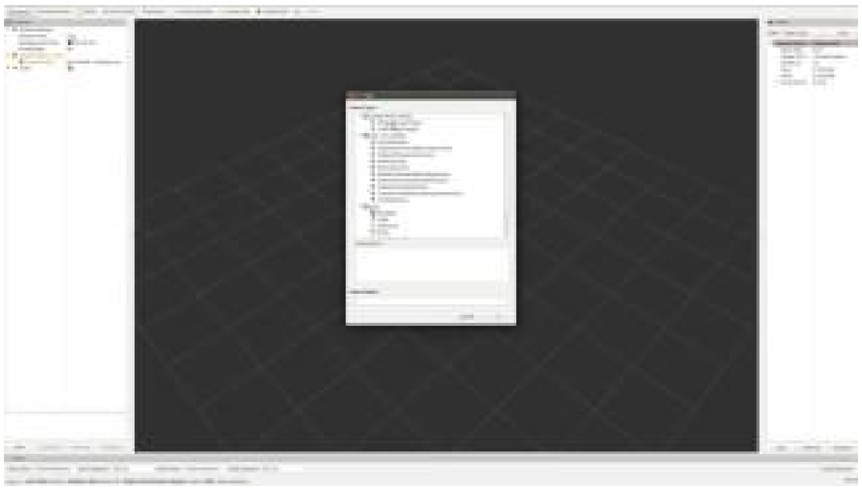
1.3.7 Detection with Yolo2



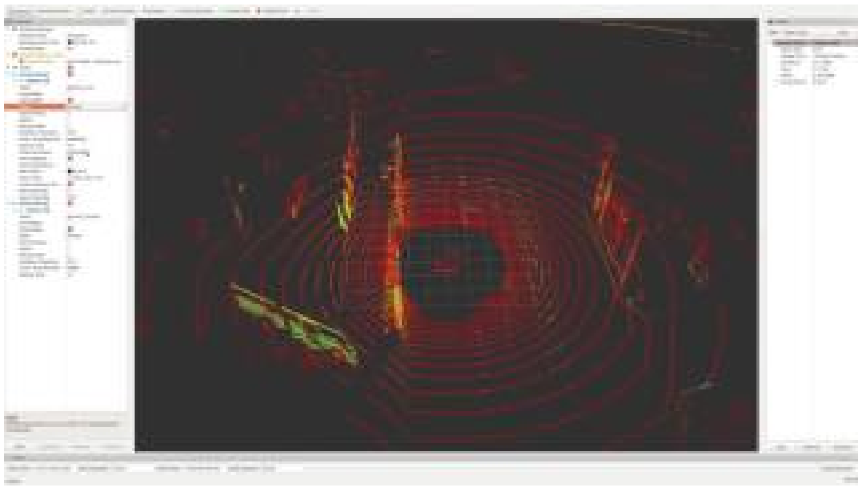
1.3.8 Detection with Yolo3



1.3.9 Detection with DPM

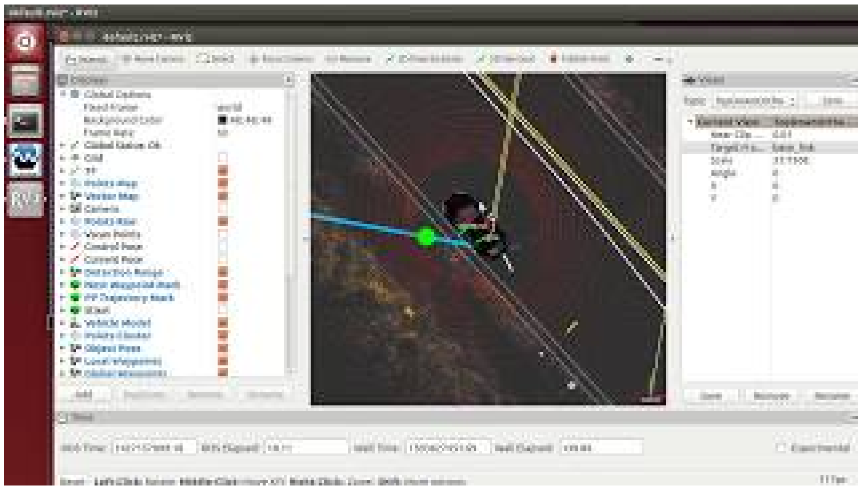


1.3.10 Detection with Euclidean Clustering

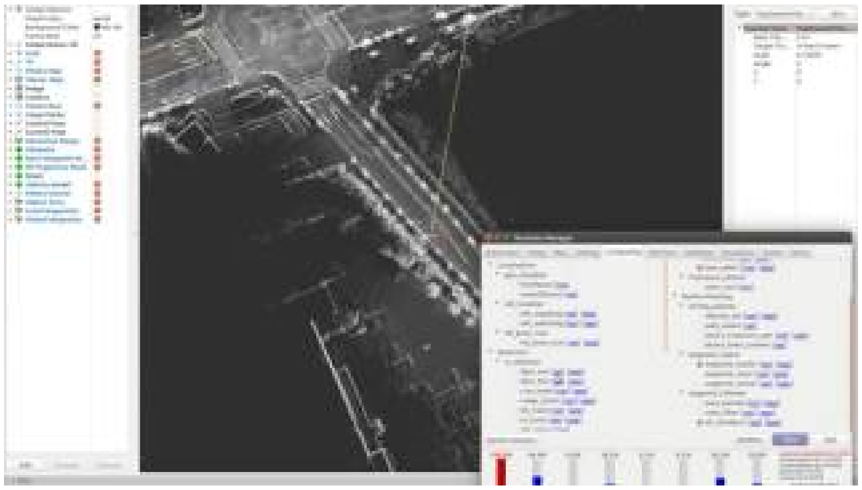


1.3.11 Traffic Light Recognition

1.3.12 Planning with ROSBAG

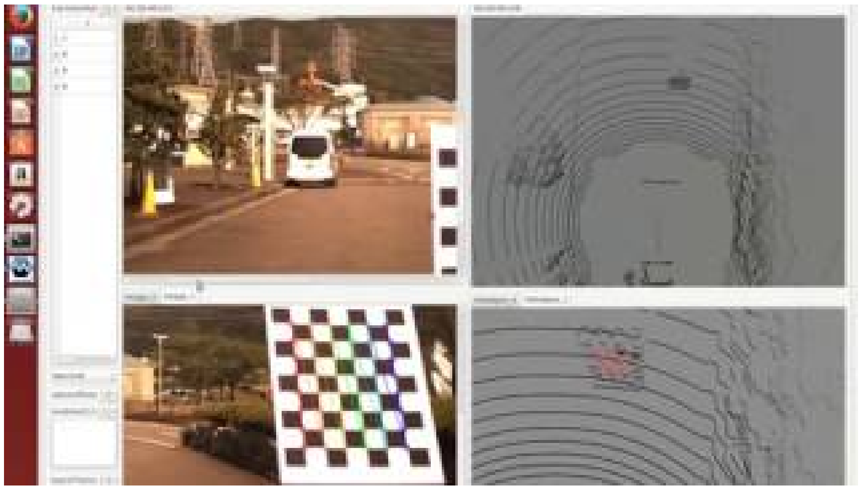


1.3.13 Planning with wf_simulator



1.3.14 Planning with Hybrid State A*

1.3.15 Calibration Toolkit



See https://github.com/CPFL/Autoware/wiki/Calibration(EN)

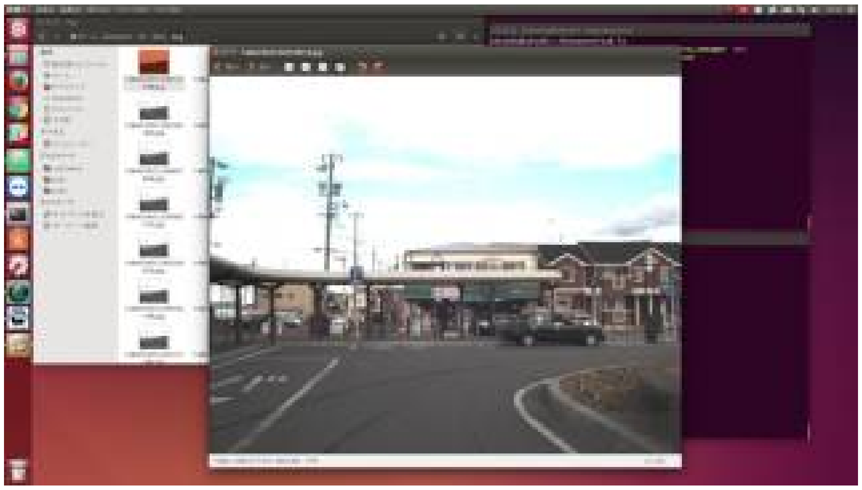
1.3.16 Camera-LiDAR Calibration

See Autoware Camera-LiDAR Calibration

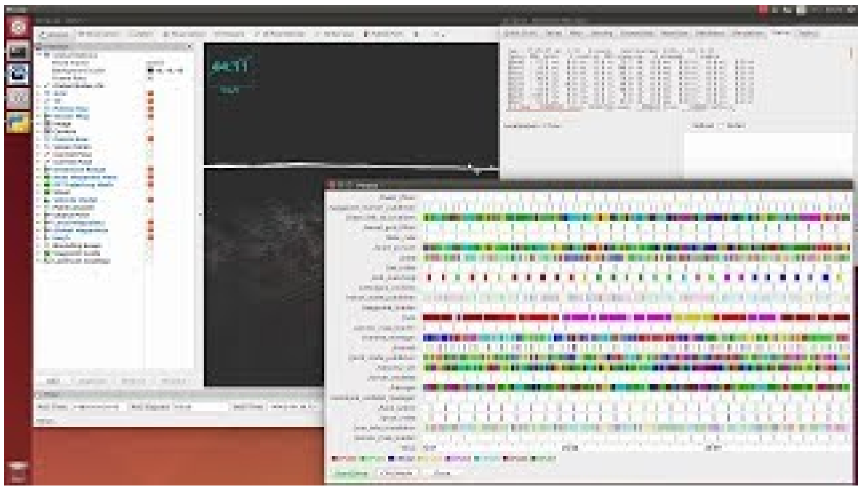
1.3.17 Multi-LiDAR Calibration

See Autoware Multi-LiDAR Calibration

1.3.18 Data Processor for Bag File



1.3.19 Ftrace



CHAPTER 2

Developers Guide

2.1 Development Process

2.1.1 Contribution Rules

You must follow the rules described below as part of the Autoware community. Please read the following carefully before you start contributing to Autoware. Thank you for your time.

To Begin With

Please join Autoware Slack at https://autoware-developer.slack.com/ and say hello to the community. You can also post your message on Autoware Googlegroups at autoware@googlegroups.com. If you want to subscribe to it, please click the "Apply to Join Group" button at https://groups.google.com/d/forum/autoware. For those who do not have Google accounts, please send an email directly to autoware+subscribe@googlegroups.com. Finally, if you do not have a GitHub account, you can create one at https://github.com/join.

Development Workflow

To assure traceability in your code, please follow our development process:

- If you are working on a feature/modify-node, create an issue first.
- Create a branch, work on the feature/modify-node, refer in at least one commit to the issue #number.
- Open a pull request for the given feature/modify-node and fill out the pull request template.

This will assure that you know where the feature originated from and the implemented code will be linked to the feature request (or bug report). Otherwise there is absolutely no traceability. Secondly, with the pull request template it will be easier for the reviewer(s) to understand the changes and we can later on convert "Steps to Reproduce" into integration tests.

Branching Model

In order to make the development process efficient, we ask you to comply with the branching model described below. On this model, we mainly use six branches - master, develop, feature, release, experimental, and hotfix.

master

This is the latest stable version of Autoware.

develop and feature

In general, developers should NOT work on the "develop" branch. When you develop new functions, please check out a new branch, "feature/[branch_name]", from the "develop" branch, and modify the code there. After the modificaiton, please send a pull request to the "develop" branch.

release

This situation is true of only the persons in charge of releasing Autoware. Once we complete some major development, we make a new release. For the release work, please checkout a new branch, "release/[1.xx.yy], from the "develop" branch, and modify the code there. After the modification, please send a pull request: "xx" in version of release/[1.xx.yy] is increased when checked out from the "develop" branch, and yy is also increased when bug fixes are done. Finally, we merge this branch to the master branch, attaching a version tag.

experimental

If your contribution is not a new feature but is to change one of the existing branches, please checkout a new branch, "experimental/[branch_name]", from the corrensponding branch. Please discuss with other contributions if this change can be merged to the original branch.

hotfix

If we encounter bugs in the "master" branch after the release, we check out the "hotfix" branch from the "master" branch and fix the bugs there. This branch is merged to each corresponding branch - master, release, and develop. After the merge, the version of the master and the release branches is increased.

Reference for the git-flow model

- http://nvie.com/posts/a-successful-git-branching-model/

Pull Request

When you are ready to send a pull request from your branch, please follow:

1. A design article should be posted with a GitHub comment for every feature or bug.
2. Every feature/bug implementation needs to be thoroughly reviewed (at least two reviewers). You can specify your favorite or appropriate reviewers by @accountname.
3. A sample program for the unit test needs to be submitted so that the reviewers or others can check if the implementation logic is correct.

4. The integration test with the demo data needs to be passed.
5. Coding style enforcement must be applied: e.g., cpplint.
6. The reviewers would further run static code analysis: e.g., cppcheck.

We introduce Travis CI to automate the above test and deploy steps.

Coding Standards

The following are regarding coding standards preferred in Autoware. We know that you must have your own coding style, but please respect our standards in our community. No need to throw away your coding style, just do your best to follow our standards.

ROS C++ Coding

First of all, please understand the ROS coding standards before you add any new code to Autoware.

- ROS Developers Guide
- ROS C++ Coding Style

You might be interested in using ROS clang-format that helps you to comply with the ROS C++ coding standards automatically in terms of styles, such as indent size and brackets space.

How to use clang-format

- Install clang-format. A newer version is better. Ubuntu has a package:

```
sudo apt-get install clang-format-x.x
```

- Locate the .clang-format file at the top directory.
- Apply clang-format to the target source file:

```
clang-format -i filename
```

Be careful that `clang-format -i` will overwrite the file. It is safe to do "git commit" before playing with clang-format. If you want to apply clang-format to the entire system, run the following script:

```
for file in $(git ls-files | \grep -E '\.(c|cpp|h|hpp)$' | \grep -v -- '#')
do
    clang-format -i $file
done
```

ROS Python Coding

In addition to C++, you will often use Python in ROS. There is also a coding style guide for Python recommended in ROS.

- ROS Python Coding Style

You can use pep8 as a tool to check PEP8-compliant coding. Therefore many existing ROS programs that use Python 2.5, but Ubuntu 16.04 or later versions will use Python 3 by default. Considering maintenance of coding in the future, thus, Python 3 is preferred in Autoware.

Notes for Package and Library Development

- Algorithms should be implemented in libraries as much as possible. For example, the normal distributions transform (NDT) algorithm could be provided as something like libndt_xxx. Library distribution allows this algorithm module to be used for other applications than ROS or Autoware. This is a spirit of open source.
- Do not make unnecessary dependencies among libraries. In particular, never make circular dependencies. This jeopardizes the entire build system.
- Do not include header files generated from msg files of other packages.
- Keep every library independent and general. Creating too many libraries is also a bad idea.
- Provide a sample program to test the functions of library.

Notes for Design and Implementation

Global Variables

You should not use global variables unless they are really needed. Instead, you should use classes or structs to hold variables. Even for libraries, you do not recommend using global variables. In C++, you can use methods. In C, you can use pointers or references for function arguments.

Besides in using global variables, you should take care of thread-safe implementation for multi-threaded programs as global variables may be accessed simultaneously among threads. In ROS, particularly, there are many other threads running in background (e.g., polling threads for subscribing to topics). Thus, you should avoid using global variable as much as possible, though you can use mutual exclusion to ensure thread-safe implementation if you really need global variables.

Arguments and Return Values

Function calls without arguments or without return values (i.e., void types) are difficult to test, because the results of function calls are all indirect and not visible from the function callees. Therefore you should make functions declared with specific arguments and meaningful return values so that a unique set of arguments always leads to the same result.

Naming

Function names must represent what these functions do. For example, `init()` or `destroy()` is not an appropriate name, because they do not tell what they initialize or destroy. Such a short and simple function name may also likely cause symbol name conflicts among multiple libraries. Function naming should be discussed when new libraries are added to Autoware. The following are some tips to solve this function naming problem.

1. Use a library name as prefix. For example, if the `fusion` library wants to export `init()` or `destroy()`, they should be named as `fusion_init()` or `fusion_destroy()`.
2. Use namespace. You can wrap the entire code of the `fusion` library by `namespace autoware::fusion {}`. This way, you can identify these functions by `autoware::fusion::init()` or `autoware::fusion_init()`. In fact, Autoware is desired to identify all the libraries, packages, and topics by namespace so that partial pieces of Autoware can be used safely in other projects.

Export Symbols

You should clarify what symbols are exported, and should not export those that would not be used or referenced by other packages and libraries. If you want not to export symbols, please use unnamed namespace or private members in classes in C++. In C, whereas, please use `static` that protects the corresponding symbols in local files.

Notes for Timing Constraints

- Do not publish topics in random periods.
- Basically, topics must be published once updated. That is to say, you should publish topics in callback functions. The following is a bad example of coding.

```
while(ros::ok()) {
    publisher.publish(xxx);
    loop.sleep();
}
```

- If a node has two or more topics, it has to publish them timely when all of them are ready. For example, if you subscribe to A and B topics, do not publish in the callback function associated with A, where only A is updated. You should wait for both A and B to be updated. The following is sample code:

```
A_callback(A_msg) {
    if (is_b_callback == true) { // If A was updated
        publish(xxx); // publish the topic
        is_a_callback = false;
        is_b_callback = false;
        return;
    }
    is_a_callback = true;
}
B_callback(B_msg) {
    if (is_a_callback == true){
        publish(xxx);
        is_a_callback = false;
        is_b_callback = false;
        return;
    }
    is_b_callback = true;
}
```

- Always put a header in the topic, and inherit the time stamp from the preceding topic. Do not update the header's time stamp without inheritance. If a node has two or more topics, you can inherit the time stamp from any of these, because their time stamps are supposed to be synchronized.
- Do not use both "service" and "topic" at the same time. If they co-exist, timing estimates become more difficult. In most cases, you should use "topic" rather than "service". However, you may use "service" for utility and interface packages, which do not require real-time performance unlike perception, planning, and control packages.
- Do not use "topic" for the large size of data, but use "nodelet" in this case. Large topic data, such as images and pointcloud scans, would sacrifice a few milliseconds to serialize and deserialize.
- Do not use "MultiThreadSpin". It is not preferable from the point of view of real-time scheduling, because timing estimates and resource allocation become more difficult.
- Do not use `output="screen"`. It is okay for the debugging purpose, however, please remove `output="screen"` before you commit to the "develop" branch - you never want to annoy your colleagues

by flooding terminal information. To monitor information, basically, we prefer ROS_INFO and ROS_DEBUG to rqt, but rqt is definitely useful for the debugging purpose. So you can use it, but just be noted that you should remove it before you commit to the "develop" branch.

- Avoid using "tf" as much as possible. For example, you can obtain the local position from the "current_pose" topic, and do not really need to use "tf". In fact, the "tf" library and ROS are disjoint (very often used together, though). Using "tf" makes timing estimates more difficult. Instead of "tf", use "topic" as much as possible. Frankly speaking, "tf" is useful for applications such as arm robots with many joints, which require dynamic transformation of coordinates, but is not very useful for self-driving vehicles because transformation of coordinates can be often statically determined.

Notes for Embedded Platforms

- Do not use a wide variety of libraries. It will decrease portability of RTOS. For example, use ros::WallTime rather than the chrono library. However, what about the boost library? It remains as an open question…
- For function arguments, use pointers and const calls by reference as much as possible. It is not necessary to use them for int or double arguments, but for vector or array arguments, you should use const calls by reference. It saves memory footprint, and also reduces overhead of the function call.
- Use the reference argument when you return from the function. A direct return value will degrade performance. However, be careful about the scope of pointer and so on. Basically, you may want to use a direct return value just for error numbers or Boolean results.
- Avoid dynamic partitioning, such as malloc and new. malloc and new could cause memory leaks. In addition, they make unclear the amount of used resources.
- If the size of vector is roughly estimated, use reserve. The vector allocates memory regions twice in case of capacity shortage. It will require a large amount of time to allocate memory regions twice, you had better to use reserve so that the required memory regions can be allocated tightly in advance.
- Avoid a monster function that spans more than 50 lines. Basically, any function should be kept around 20-30 lines of code. In addition, bear in mind that the granularity of coding within the function should be well balanced. According to Effective C++, a method or function of sophisticated code has only 14 lines on average.

Bad Example

```
callback() {
    start_time = clock_get_time(); // Not abstracted at all
    compute_xxx(); // Abstracted too much
}
```

- Use inline effectively. Such a function that has a single line of code, for instance, should be inline. However, inline functions will enlarge footprint. So be careful about using too many inline functions.
- Function naming should correspond to the function code. For example, do not write heavy code in get() or set(), because these functions are supposed to just get or set some values. Function naming should imply what the function is and what is the cost of processing time. If you want to create a time-consuming function, for example, probably function naming such as compute_xxx is suitable.

C++ Books

C++ 11/14/17 has introduced many useful capabilities, e.g., type inference. You may want to review C++ 11 through the popular books: [Amazon links]

2.2 Design & Architecture

2.2.1 Overview

2.2.2 Design Rule

2.2.3 Sensing

2.2.4 Detection

2.2.5 Localization

2.2.6 Prediction

2.2.7 Decision

2.2.8 Mission Planning

2.2.9 Motion Planning

2.2.10 Actuation

2.2.11 etc. . .

2.3 Packages & Nodes

2.3.1 Sensing

Warning: Sensor drivers will be removed from Autoware.

Autoware's adi_driver Subtree

This directory is part of a subtree fetched from https://github.com/CPFL/adi_driver on the **Autoware** branch, a fork from the original https://github.com/ros-drivers/velodyne

This repo adds specific functions for Autoware. If you need to modify **any** file inside this folder structure, please use the following commands to either push or fetch changes from the subtree. All the commands written here will suppose you're in the root of Autoware path.

Pulling in commits from the repository subtree

Bring latest commits from https://github.com/CPFL/adi_driver

```
git subtree pull --prefix ros/src/sensing/drivers/imu/packages/analog_devices
https://github.com/CPFL/adi_driver Autoware --squash
```

Pushing changes to the repository subtree

If you made any modification to the subtree you are encouraged to commit and publish your changes to the fork. You can do with the following command.

```
git subtree push --prefix ros/src/sensing/drivers/imu/packages/analog_devices
https://github.com/CPFL/adi_driver Autoware
```

Original README below

adi_driver

This package contains ROS driver nodes for Analog Devices(ADI) sensor products mainly communicate by SPI(Serial Periferal Interface).

Currently supported devices are:

- ADIS16470
  - Wide Dynamic Range Mini MEMS IMU
- ADXL345:
  - 3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer
  - The support for this device is experimental

You need a SPI interface on your PC to communicate with device. This package supports Devantech's USB-IIS as the USB-SPI converter.

USB-IIS

Overview

USB-IIS is a USB to Serial/I2C/SPI converter, simple, small and easy to use. You don't need any extra library like libusb or libftdi. The device is available on /dev/ttyACM* as modem device.

Please consult the product information and SPI documentation for the detail.

Tips

You need to remove the jumper block on `Power link` pins to provide 3.3V for the device.

You need to add your user to dialout group to acces /dev/ttyACM* .

```
$ sudo adduser your_user_name dialout
```

If it takes several seconds until /dev/ttyACM* available, you need to uninstall modemmanager as:

```
$ sudo apt remove modemmanager
```

ADIS16470

Overview

ADIS16470 is a complete inertial system that includes a triaxis gyroscope and a triaxis accelerometer.

You can use Breakout board for easy use.

Connection

You need to build a flat cable to connect the USB-ISS and the ADIS16470 breakout board. The picture shows a implementation.

Very simple schematic is here. J1 is the USB-ISS pin and J2 is the 2mm pin headers on the ADIS16470 breakout board.

Note: you only need to connect one of the power-line(3.3V and GND). They are connected in the breakout board.

BOM

- J1: 2550 Connector 6pin
  - Available at Akiduki
- J2: FCI Connector for 1.0mm pitch ribon cables
  - Available at RS Components
- 1.0 mm pitch ribon cable
  - Available at Aitendo

Quick start

Connect your sensor to USB port. Run the launch file as:

```
$ roslaunch adi_driver adis16470.launch
```

You can see the model of ADIS16470 breakout board in rviz panel.

ADXL345

Overview

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ± 16g. At this moment, support for this device is experimental.

Autoware Camera-LiDAR Calibration Package

How to calibrate

Camera-LiDAR calibration is performed in two steps:

1. Obtain camera intrinsics
2. Obtain camera-LiDAR extrinsics

Camera intrinsic calibration

The intrinsics are obtained using the `autoware_camera_calibration` script, which is a fork of the official ROS calibration tool.

How to launch

1. In a sourced terminal:`rosrun autoware_camera_lidar_calibrator cameracalibrator.py --square SQUARE_SIZE --size MxN image:=/image_topic`
2. Play a rosbag or stream from a camera in the selected topic name.
3. Move the checkerboard around within the field of view of the camera until the bars turn green.
4. Press the `CALIBRATE` button.
5. The output and result of the calibration will be shown in the terminal.
6. Press the `SAVE` button.
7. A file will be saved in your home directory with the name `YYYYmmdd_HHMM_autoware_camera_calibration.yaml`.

This file will contain the intrinsic calibration to rectify the image.

Parameters available

Matlab checkerboard detection engine (beta)

This node additionally supports the Matlab engine for chessboard detection, which is faster and more robust than the OpenCV implementation.

1. Go to the Matlab python setup path `/PATH/TO/MATLAB/R201XY/extern/engines/python`.
2. Run `python setup.py install` to setup Matlab bindings.

To use this engine, add `--detection matlab` to the list of arguments, i.e.`rosrun autoware_camera_lidar_calibrator cameracalibrator.py --detection matlab --square SQUARE_SIZE --size MxN image:=/image_topic`

DevelopersGuide/PackagesAPI/sensing/docs/camera_calibration.jpg

---

Camera-LiDAR extrinsic calibration

Camera-LiDAR extrinsic calibration is performed by clicking on corresponding points in the image and the point cloud.

This node uses `clicked_point` and `screenpoint` from the `rviz` and `image_view2` packages respectively.

How to launch

1. Perform the intrinsic camera calibration using camera intrinsic calibration tool described above (resulting in the file `YYYYmmdd_HHMM_autoware_camera_calibration.yaml`).
2. In a sourced terminal:`roslaunch autoware_camera_lidar_calibrator camera_lidar_calibration.launch intrinsics_file:=/PATH/TO/YYYYmmdd_HHMM_autoware_camera_calibration.yaml image_src:=/image`
3. An image viewer will be displayed.
4. Open Rviz and show the point cloud and the correct fixed frame.
5. Observe the image and the point cloud simultaneously.
6. Find a point within the image that you can match to a corresponding point within the point cloud.
7. Click on the pixel of the point in the image.
8. Click on the corresponding 3D point in Rviz using the *Publish Point* tool.
9. Repeat this with at least 9 different points.
10. Once finished, a file will be saved in your home directory with the name `YYYYmmdd_HHMM_autoware_lidar_camera_calibration.yaml`.

This file can be used with Autoware's Calibration Publisher to publish and register the transformation between the LiDAR and camera. The file contains both the intrinsic and extrinsic parameters.

Parameters available

Camera-LiDAR calibration example

To test the calibration results, the generated yaml file can be used in the `Calibration Publisher` and then the `Points Image` in the **Sensing** tab.

DevelopersGuide/PackagesAPI/sensing/docs/camera_lidar_calibration.jpg

Notes

This calibration tool assumes that the Velodyne is installed with the default order of axes for the Velodyne sensor.

- X axis points to the front
- Y axis points to the left
- Z axis points upwards

Baumer Package

This package allows the capture of an image stream from Baumer cameras. It was tested successfully using a VLG22C.

Requirements

Baumer SDK installed in HOME (Default path)

How to launch

- From a sourced terminal:

```
roslaunch vlg22c_cam baumer.launch
```

- From Runtime Manager:

Sensing Tab -> Cameras -> VLG-22

Parameters

Launch file available parameters:

Notes

- The SDK needs to be obtained from Baumer's website.
- The node will only be compiled if the SDK is installed in the default directory inside ${HOME}

Calibration Publisher

This nodes publishes the camera intrinsics, extrinsics and registers the TF between the camera and LiDAR sensors. The data is read from an Autoware compatible calibration file.

Publications

- `sensor_msgs/CameraInfo`, default topic name `/NAMESPACE/camera_info`.
- `autoware_msgs/projection_matrix`, default topic name `/NAMESPACE/camera_info`

`NAMESPACE` if any.

Subscriptions

- `sensor_msgs/Image`, default topic name `/image_raw`

Parameters

Compare Map Filter

Autoware package that compare the LiDAR PointCloud and PointCloud Map and then extract (or eliminate) coincident points.

Requirements

- PointCloud Map with extremely few unnecessary PointCloud (people, cars, etc.).

How to launch

- From a sourced terminal:
  - `roslaunch points_preprocessor compare_map_filter.launch`
- From Runtime Manager:

Sensing Tab -> Points Preprocessor -> `compare_map_filter` You can change the config, as well as other parameters, by clicking [app]

Parameters

Launch file available parameters:

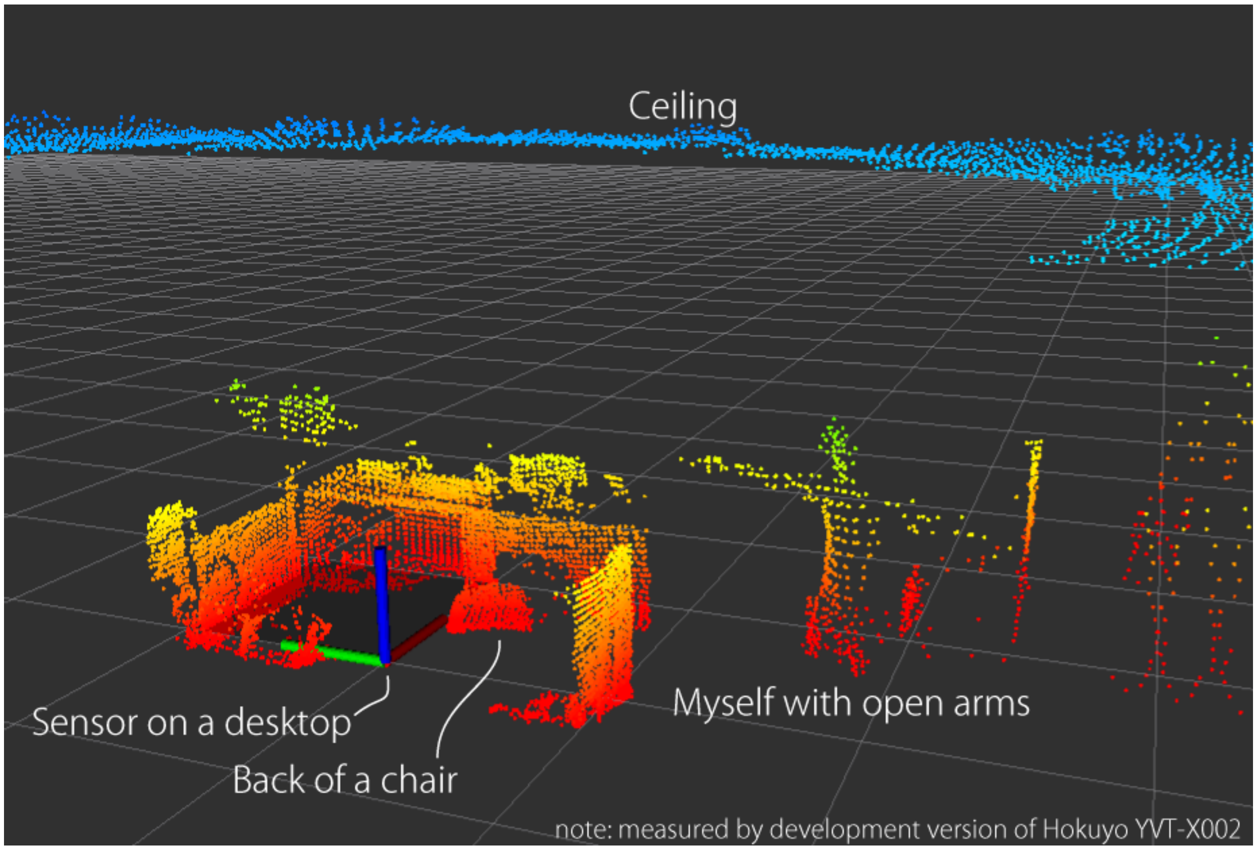
Subscribed topics

Published topics

Video

ROS driver for HOKUYO 3D sensor

A driver node for HOKUYO's new 3D sensor, which will be released in 2015.



Autoware's sick_ldmrs_laser Subtree

This directory is part of a subtree fetched from https://github.com/CPFL/sick_ldmrs_laser on the **Autoware** branch, a fork from the original https://github.com/SICKAG/sick_ldmrs_laser

This repo adds specific functions for Autoware. If you need to modify **any** file inside this folder structure, please use the following commands to either push or fetch changes from the subtree. All the commands written here will suppose you're in the root of Autoware path.

Pulling in commits from the repository subtree

Bring latest commits from https://github.com/CPFL/sick_ldmrs_laser

```
git subtree pull --prefix ros/src/sensing/drivers/lidar/packages/sick/ldmrs/
sick_ldmrs_laser https://github.com/CPFL/sick_ldmrs_laser Autoware --squash
```

Pushing changes to the repository subtree

If you made any modification to the subtree you are encouraged to commit and publish your changes to the fork. You can do with the following command.

```
git subtree push --prefix ros/src/sensing/drivers/lidar/packages/sick/ldmrs/
sick_ldmrs_laser https://github.com/CPFL/sick_ldmrs_laser Autoware
```

Original README below

sick_ldmrs_laser

This stack provides a ROS driver for the SICK LD-MRS series of laser scanners. The SICK LD-MRS is a multi-layer, multi-echo 3D laser scanner that is geared towards rough outdoor environments and also provides object tracking. The driver also works for the identical devices from IBEO.

Supported Hardware

This driver should work with all of the following products. However, it has only been tested and confirmed working on the LD-MRS800001S01 so far. If you try any of the other scanners and run into trouble, please open an issue.

Installation

In the following instructions, replace `<rosdistro>` with the name of your ROS distro (e.g., `indigo`).

From binaries

The driver has not been released yet. But once that happens, you can install it directly by typing:

`~~sudo apt-get install ros-<rosdistro>-sick-ldmrs-laser~~`

From source

```
source /opt/ros/<rosdistro>/setup.bash
mkdir -p ~/ros_catkin_ws/src/
cd ~/ros_catkin_ws/src/
git clone https://github.com/SICKAG/libsick_ldmrs.git
git clone -b <rosdistro> https://github.com/SICKAG/sick_ldmrs_laser.git
cd ..
catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release
source ~/ros_catkin_ws/install_isolated/setup.bash
```

Quick Start

```
roslaunch sick_ldmrs_tools sick_ldmrs_demo.launch
rosrun rviz rviz -d $(rospack find sick_ldmrs_tools)/config/sick_ldmrs.rviz
rosrun rqt_reconfigure rqt_reconfigure
rosrun rqt_robot_monitor rqt_robot_monitor
```

Package overview

- **sick_ldmrs_driver**: This package provides the main driver node, `sick_ldmrs_node`.
- **sick_ldmrs_description**: This package provides an URDF description of the scanner along with the corresponding mesh.

- **sick_ldmrs_msgs**: This package provides ROS message definitions for the scanner's object tracking functionality and a PCL point type used in the PointCloud2 topic.
- **sick_ldmrs_tools**: This package provides example nodes that demonstrate how to subscribe to the topics provided by the scanner and use the data:
  - `sick_ldmrs_filter_first`: Filters the point cloud and only output the first echo of each beam.
  - `sick_ldmrs_filter_last`: Filters the point cloud and only output the last echo of each beam.
  - `sick_ldmrs_filter_layer`: Filters the point cloud and only output points from the second layer.
  - `sick_ldmrs_object_marker`: Subscribes to the `objects` topic and publishes the object tracking information as visualization_msgs/MarkerArray for visualization in RViz.
  - `sick_ldmrs_all_layer_assembler`: Subscribes to the point cloud, transforms it into a fixed frame and assembles one point clouds of the lower four layers and one of the upper four layers into a point cloud of all eight layers.
  - `sick_ldmrs_make_organized`: Subscribes to the point cloud and turns it into an organized point cloud.
  - `sick_ldmrs_remove_background`: Removes static background points from an organized input point cloud, assuming the scanner is stationary.
  - `sick_ldmrs_print_resolution`: Subscribes to the point cloud and prints the angular resolution sectors. Useful to verify that the FocusedRes / FlexRes features are working as expected.

ROS API

sick_ldmrs_node

Published Topics

`cloud` (sensor_msgs/PointCloud2)

- The published point cloud. The meaning of the fields is documented in sick_ldmrs_point_type.h.

`objects` (sick_ldmrs_msgs/ObjectArray)

- The output of the object tracking functionality of the scanner. See ObjectArray.msg and Object.msg.

`diagnostics` (diagnostic_msgs/DiagnosticArray)

- ROS diagnostics information.

Parameters

Dynamically Reconfigurable Parameters

See the dynamic_reconfigure package for details on dynamically reconfigurable parameters.

`~frame_id` (`str`, default: ldmrs)

- The TF frame in which point clouds will be returned.

`~start_angle` (`double`, default: 0.872664625997)

- The angle of the first range measurement [rad]. Range: -1.04610672041 to 0.872664625997

`~end_angle` (`double`, default: -1.0471975512)

- The angle of the last range measurement [rad]. Range: -1.0471975512 to 0.871573795215

`~scan_frequency` (`int`, default: 0)

- Scan frequency. Possible values are: ScanFreq1250 (0): Scan frequency 12.5 Hz, ScanFreq2500 (1): Scan frequency 25.0 Hz, ScanFreq5000 (2): Scan frequency 50.0 Hz.

`~sync_angle_offset` (`double`, default: 0.0)

- Angle under which the LD-MRS measures at the time of the sync pulse [rad]. Range: -3.14159265359 to 3.1410472382

`~angular_resolution_type` (`int`, default: 1)

- Angular resolution type. Possible values are: FocusedRes (0): Focused resolution, ConstantRes (1): Constant resolution (0.25° @ 12.5 Hz / 0.25° @ 25.0 Hz / 0.5° @ 50.0 Hz), FlexRes (2): Flexible resolution

`~layer_range_reduction` (`int`, default: 0)

- Possible values are: RangeFull (0): All layers full range, RangeLowerReduced (1): Lower 4 layers reduced range, RangeUpperReduced (2): Upper 4 layers reduced range, RangeAllReduced (3): All 8 layers reduced range.

`~ignore_near_range` (`bool`, default: False)

- Ignore scan points up to 15m. Requires `layer_range_reduction` = RangeLowerReduced.

`~sensitivity_control` (`bool`, default: False)

- Reduce the sensitivity automatically in case of extraneous light.

`~flexres_start_angle1` (`double`, default: 0.872664625997)

- FlexRes: start angle of sector 1. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle2` (`double`, default: 0.610865238198)

- FlexRes: start angle of sector 2. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle3` (`double`, default: 0.523598775598)

- FlexRes: start angle of sector 3. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle4` (`double`, default: 0.349065850399)

- FlexRes: start angle of sector 4. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle5` (`double`, default: 0.0)

- FlexRes: start angle of sector 5. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle6` (`double`, default: -0.349065850399)

- FlexRes: start angle of sector 6. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle7` (`double`, default: -0.523598775598)

- FlexRes: start angle of sector 7. Range: -1.04610672041 to 0.872664625997

`~flexres_start_angle8` (`double`, default: -0.698131700798)

- FlexRes: start angle of sector 8. Range: -1.04610672041 to 0.872664625997

`~flexres_resolution1` (`int`, default: 32)

- FlexRes: angular resolution of sector 1. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution2` (`int`, default: 16)

- FlexRes: angular resolution of sector 2. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution3` (`int`, default: 8)

- FlexRes: angular resolution of sector 3. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution4` (`int`, default: 4)

- FlexRes: angular resolution of sector 4. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution5` (`int`, default: 8)

- FlexRes: angular resolution of sector 5. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution6` (`int`, default: 16)

- FlexRes: angular resolution of sector 6. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution7` (`int`, default: 32)

- FlexRes: angular resolution of sector 7. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~flexres_resolution8` (`int`, default: 16)

- FlexRes: angular resolution of sector 8. Possible values are: Res0125 (4): Angular resolution 0.125 degrees, Res0250 (8): Angular resolution 0.25 degrees, Res0500 (16): Angular resolution 0.5 degrees, Res1000 (32): Angular resolution 1.0 degrees

`~contour_point_density` (`int`, default: 2)

- Contour point density. Possible values are: ClosestPointOnly (0): Closest point only, LowDensity (1): Low density, HighDensity(2): High density

`~min_object_age` (`int`, default: 0)

- Minimum tracking age (number of scans) of an object to be transmitted. Range: 0 to 65535

`~max_prediction_age` (`int`, default: 0)

- Maximum prediction age (number of scans) of an object to be transmitted. Range: 0 to 65535

Not Dynamically Reconfigurable Parameters

`~hostname` (`string`, default: "192.168.0.1")

- The host name or IP address of the laser scanner.

Notes on FlexRes

Please observe the following constraints when setting the FlexRes parameters (user-defined sectors of angular resolution):

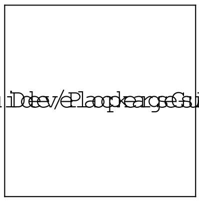
- The start angles of each sector have to be given in decreasing order (i.e., `flexres_start_angle1 > flexres_start_angle2` etc.)
- The sectors with a resolution of 0.125° must not sum up to more than 20°.
- The number of shots per scan must be at most 440. (This corresponds to an average angular resolution of 0.25° over the full 110° range.)

The reason for the last two constraints is that the scanner increases the shooting frequency when increasing angular resolution. To avoid overheating the scanner and limit the amount of data to be processed, it's not possible to use 0.125° angular resolution for the full range.

Unused Scanner Parameters

The scanner provides a number of parameters that are not exposed via the ROS API. Specifically:

- A number of vehicle-related parameters (vehicle velocity, axis lengths, mounting position of the scanner). The "ROS way" of handling this is to use URDF to specify the position of the scanner and other forms of localization to provide the transform between world and scanner coordinates.
- The `upside_down` parameter. In ROS, this is also better handled by specifying the scanner mounting position in the URDF. This driver will print a warning if the `upside_down` parameter was set externally.
- SOPAS fields and eval cases.
- *Setting* the TCP/IP configuration of the scanner. Not supported by this driver to avoid accidentally making the scanner unaccessible; use the software provided with the scanner instead.

 `DevelopersGu`[illegible]`_eng_sm.png`

Lidar Sick Package

Steps to Execute the Package

Using Runtime Manager

1. Go to the `Sensing` Tab.
2. Click on the `config` button next to the SICK LMS511 label.
3. Write the current IP address of the SICK LMS511 that you wish to connect.
4. Launch the node, clicking on the checkbox.

Using the command line

In a sourced terminal, execute `roslaunch sick_driver lms511.launch ip:=XXX.YYY.ZZZ.WWW`.

To confirm that data is coming from the LiDAR

1. Open RViz.
2. Change the fixed frame to `sick`.
3. Add topic type Scan, with the name `/scan`.

microstrain_3dm_gx5_45

This package is a modification of microstrain_3dm_gx5_45 package.

Original source code is here.

Description

Interface software, including ROS node, for Microstrain 3DM-GX5-45.

See http://wiki.ros.org/microstrain_3dm_gx5_45

Multi LiDAR Calibrator

This package allows to obtain the extrinsic calibration between two PointClouds with the help of the NDT algorithm.

The `multi_lidar_calibrator` node receives two `PointCloud2` messages (parent and child), and an initialization pose. If possible, the transformation required to transform the child to the parent point cloud is calculated, and output to the terminal.

How to launch

1. **You'll need to provide an initial guess, otherwise the transformation won't converge.**
2. In a sourced terminal:

Using rosrun

```
rosrun multi_lidar_calibrator multi_lidar_calibrator _points_child_src:=/
lidar_child/points_raw _points_parent_src:=/lidar_parent/points_raw _x:=0.0
_y:=0.0 _z:=0.0 _roll:=0.0 _pitch:=0.0 _yaw:=0.0
```

Using roslaunch

```
roslaunch multi_lidar_calibrator multi_lidar_calibrator points_child_src:=/
lidar_child/points_raw points_parent_src:=/lidar_parent/points_raw x:=0.0
y:=0.0 z:=0.0 roll:=0.0 pitch:=0.0 yaw:=0.0
```

1. Play a rosbag with both lidar data `/lidar_child/points_raw` and `/lidar_parent/points_raw`
2. The resulting transformation will be shown in the terminal as shown in the *Output* section.
3. Open RViz and set the fixed frame to the Parent

4. Add both point cloud `/lidar_parent/points_raw` and `/points_calibrated`
5. If the algorithm converged, both PointClouds will be shown in rviz.

Input topics

Output

1. Child Point cloud transformed to the Parent frame and published in `/points_calibrated`.
2. Output in the terminal showing the X,Y,Z,Yaw,Pitch,Roll transformation between child and parent. These values can be used later with the `static_transform_publisher`.

Output example:

```
transformation from ChildFrame to ParentFrame
This transformation can be replicated using:

rosrun tf static_transform_publisher 1.7096 -0.101048 -0.56108 1.5708 0.00830573  0.
↪843 /ParentFrame /ChildFrame 10
```

The figure below shows two lidar sensors calibrated by this node. One is shown in gray while the other is show in blue. Image obtained from rviz.

DevelopersGuide/PackagesAPI/sensing/doc/calibration_result.jpg

Autoware Point Grey Camera Drivers Package

This package allows the capture of an image stream from Point Grey cameras. It has been tested successfully with Grasshopper3 and LadyBug5 devices on both Ubuntu 14.04 and 16.04.

Requirements

- FlyCapture SDK provided by Point Grey.

Grasshopper3

How to launch

- From a sourced terminal:`roslaunch autoware_pointgrey_drivers grasshopper3.launch`
- From Runtime manager:Sensing Tab -> Cameras -> PointGrey Grasshopper3

Parameters available

Ladybug

How to launch

- From a sourced terminal:`roslaunch autoware_pointgrey_drivers ladybug.launch`
- From Runtime manager:Sensing Tab -> Cameras -> PointGrey Ladybug5

Parameters available

Notes

- The FlyCapture SDK must be obtained from Point Grey's website.https://www.ptgrey.com/flycapture-sdk

Some Guidelines for Tuning the Ring Ground Filter

Author : Patiphon Narksri

Problem: Some parts of vertical objects are detected as ground points.

FIX: One workaround for this is to set the "clipping_threshold" parameter in the launch file. By setting this threshold, any points higher than this threshold will be detected as vertical points. However, because all points that are higher than the threshold will be detected as vertical points, slopes might be incorrectly detected as vertical points as well.

Problem: Some small objects like curbs are missing. *FIX:* Try to lower the "gap_thres" parameter. However, lowering this parameter too much might result in mis-detecting slopes as vertical points. Usually, 0.15 - 0.2 is enough for detecting curbs.

Problem: Ring shaped noise (ground points being detected as vertical points) occurs nearby the vehicle. *FIX:* Try to lower the "points_distance" parameter. However, lowering this parameter too much might result in mis-detecting vertical objects which are far away from the vehicle as ground points.

Problem: Line shaped noise (in radial direction) occurs near edges of vertical objects. *FIX:* Decrease the "min_points" parameter. However, by doing so, some parts of vertical objects will be mis-detected as ground points.

1. Prerequisites

(1) Install a ubuntu PC. We suggested Ubuntu 14.04 or Ubuntu 16.04. Please do not use virtual machine. (2) Install ros full-desktop version. We tried Indigo and Kinect. (3) Please also install libpcap-dev.

2. Install

(1). Copy the whole rslidar ROS driver directory into ROS workspace, i.e "~/catkin_ws/src".

(2). Check the file attributes:

```
cd ~/catkin_ws/src/ros_rslidar/rslidar_drvier
chmod 777 cfg/*
cd ~/catkin_ws/src/ros_rslidar/rslidar_pointcloud
chmod 777 cfg/*
```

(3). Then to compile the source code and to install it:

```
cd ~/catkin_ws
catkin_make
```

3. Configure PC IP

By default, the RSLIDAR is configured to **192.168.1.200** as its device IP and **192.168.1.102** as PC IP that it would communicate. The default **MSOP port is 6699** while **DIFOP port is 7788**. So you need configure your PC IP as a static one **192.168.1.102**.

4. Run as independent node

We have provide example launch files under rslidar_pointcloud/launch, we can run the launch file to view the point cloud data. For example, if we want to view RS-LiDAR-16 real time data: (1). Open a new terminal and run:

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch rslidar_pointcloud rs_lidar_16.launch
```

(2). Open a new terminal and run:

```
rviz
```

Set the Fixed Frame to "**rslidar**". Add a Pointcloud2 type and set the topic to "**rslidar_points**".

5. Run as nodelet

We can also run the driver node and cloud node as a nodelet. Open a new terminal and run:

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch rslidar_pointcloud cloud_nodelet_16.launch
# or
roslaunch rslidar_pointcloud cloud_nodelet_32.launch
```

Then we can run view the pointcloud via "rviz"

6. About the lidar calibration parameters

Under "**rslidar_pointcloud/data**" directory, you can find the lidar calibration parameters files for the exact sensor. By default the launch file "rs_lidar_16.launch" load the three files:

- rslidar_pointcloud/data/rs_lidar_16/angle.csv
- rslidar_pointcloud/data/rs_lidar_16/ChannelNum.csv
- rslidar_pointcloud/data/rs_lidar_16/curves.csv.

If you have more than one RSLIDAR, you can create new sub-directories under the "**rslidar_pointcloud/data/**", and put the data files into it.Then you need rewrite the launch file to start you lidar. We have put an example launch file "two_lidar.launch" to load two lidars together for reference.

Begin from 2018.09.01 In the launch file, we could set the MSOP port and DIFOP port. MSOP port is used for receive the main point cloud data, while DIFOP port is used for receive the device information data. If we set the right DIFOP port, we will get the lidar calibration parameters from the DIFOP packets not from the files, so can ignore the local lidar calibration files. About how to check the DIFOP port, please reference the Appendix G in RS-LiDARR manual.

Launch file for nmea_navsat_driver

Serial port reader and parser for NMEA compatible GPS devices.

Setup

```
$ sudo apt-get install ros-indigo-nmea-navsat-driver
```

ROS run

```
$ rosrun nmea_navsat_driver nmea_serial_driver _port:=/dev/ttyUSB0 _baud:=57600
```

Autoware's Velodyne Driver Subtree

This directory is part of a subtree fetched from https://github.com/CPFL/velodyne on the **Autoware** branch, a fork from the original https://github.com/ros-drivers/velodyne

This repo adds support to HDL-64 S3 and creates the launch files used by Autoware. If you need to modify **any** file inside this folder structure, please use the following commands to either push or fetch changes from the subtree. All the commands written here will suppose you're in the root of Autoware path.

Pulling in commits from the repository subtree

Bring latest commits from https://github.com/CPFL/velodyne

```
git subtree pull --prefix ros/src/sensing/drivers/lidar/packages/velodyne
https://github.com/CPFL/velodyne Autoware --squash
```

Pushing changes to the repository subtree

If you made any modification to the subtree you are encouraged to commit and publish your changes to the fork. You can do with the following command.

```
git subtree push --prefix ros/src/sensing/drivers/lidar/packages/velodyne
https://github.com/CPFL/velodyne Autoware
```

End of Section

Original README from https://github.com/ros-drivers/velodyne

Overview

Velodyne_ is a collection of ROS_ packages supporting `Velodyne high definition 3D LIDARs_`.

Warning::

The master branch normally contains code being tested for the next ROS release. It will not always work with every previous release.

The current `master` branch works with ROS Kinetic, Jade, and Indigo. It may work with Hydro and Groovy, but that has not been tested recently. To build for Fuerte from source, check out the `rosbuild` branch instead of `master`.

.. _ROS: http://www.ros.org .. _Velodyne: http://www.ros.org/wiki/velodyne .. _`Velodyne high definition 3D LIDARs`: http://www.velodynelidar.com/lidar/lidar.aspx

Prerequisites

- Install the MTi USB Serial Driver

```
$ git clone https://github.com/xsens/xsens_mt.git
$ cd ~/xsens_mt
$ make
$ sudo modprobe usbserial
$ sudo insmod ./xsens_mt.ko
```

- Install gps_common

```
$ sudo apt-get install ros-distro-gps-common
```

Running the Xsens MTi ROS Node

1. Copy the contents of the src folder into your catkin workspace 'src' folder. Make sure the permissions are set to *o+rw* on your files and directories. For details on creating a catkin workspace environment refer to Creating a catkin ws
2. in your catkin_ws ($CATKIN) folder, execute

```
$ catkin_make
```

3. Source the environment for each terminal you work in. If necessary, add the line to your .bashrc

```
. $CATKIN/devel/setup.bash
```

4. Initiate the ros core

```
$ roscore
```

5. Open a new terminal, type

```
$ . $CATKIN/devel/setup.bash
$ rosrun xsens_driver mtdevice.py -m $sm -f $fs # publish sensor data
```

where $fs can be 1,5,10,20,40,50,80,100,200 or 400Hz. This configures the MTi to output inertial data and magnetometer data at the set ODR. The maximum supported inertial update rate is 400Hz and for the magnetometer it is 100Hz. The $sm can be set to 1,2 or 3. This can be used to set the sensor to output sensor data or filter outputs.

6. To run the node

```
$ rosrun xsens_driver mtnode.py _device:=/dev/ttyUSB0 _baudrate:=115200
```

or

```
$ rosrun xsens_driver mtnode.py
```

7. Open a new terminal (do not forget step 3)

```
$ . $CATKIN/devel/setup.bash
$ rostopic echo /mti/sensor/sample
```

or

```
$ . $CATKIN/devel/setup.bash
$ rostopic echo /mti/sensor/imu
```

Troubleshooting

- The Mti1 (Motion Tracker Development Board) is not recognized. Support for the Development Board is present in recent kernels. (Since June 12, 2015).If your kernel does not support the Board, you can add this manually

  $ sudo /sbin/modprobe ftdi_sio $ echo 2639 0300 | sudo tee /sys/bus/usb-serial/drivers/ftdi_sio/new_id

- The device is recognized, but I cannot ever access the device Make sure you are in the correct group (often dialout or uucp) in order to access the device. You can test this with

```
  $ ls -l /dev/ttyUSB0
  crw-rw---- 1 root dialout 188, 0 May  6 16:21 /dev/ttyUSB0
  $ groups
  dialout audio video usb users plugdev
```

  If you aren't in the correct group, you can fix this in two ways.

  1. Add yourself to the correct group You can add yourself to it by using your distributions user management tool, or call

```
 $ sudo usermod -G dialout -a $USER
```

     Be sure to replace dialout with the actual group name if it is different. After adding yourself to the group, either relogin to your user, or call

```
 $ newgrp dialout
```

     to add the current terminal session to the group.

  2. Use udev rules Alternatively, put the following rule into /etc/udev/rules.d/99-custom.rules

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="2639", ACTION=="add", GROUP="$GROUP",␣
↪MODE="0660"
```

Change $GROUP into your desired group (e.g. adm, plugdev, or usb).

- The device is inaccessible for a while after plugging it in When having problems with the device being busy the first 20 seconds after plugin, purge the modemmanager application.

2.3.2 Detection

Lidar fake perception

This node generates fake object and pointcloud message based on the value given manually. At the sametime, real pointclouds and real detected objects can be merged as sources.

How to launch

- From Runtime Manager:

Computing -> Detection -> lidar_detector -> lidar_fake_perception

- From CLI:

```
$ roslaunch lidar_fake_perception lidar_fake_perception.launch
```

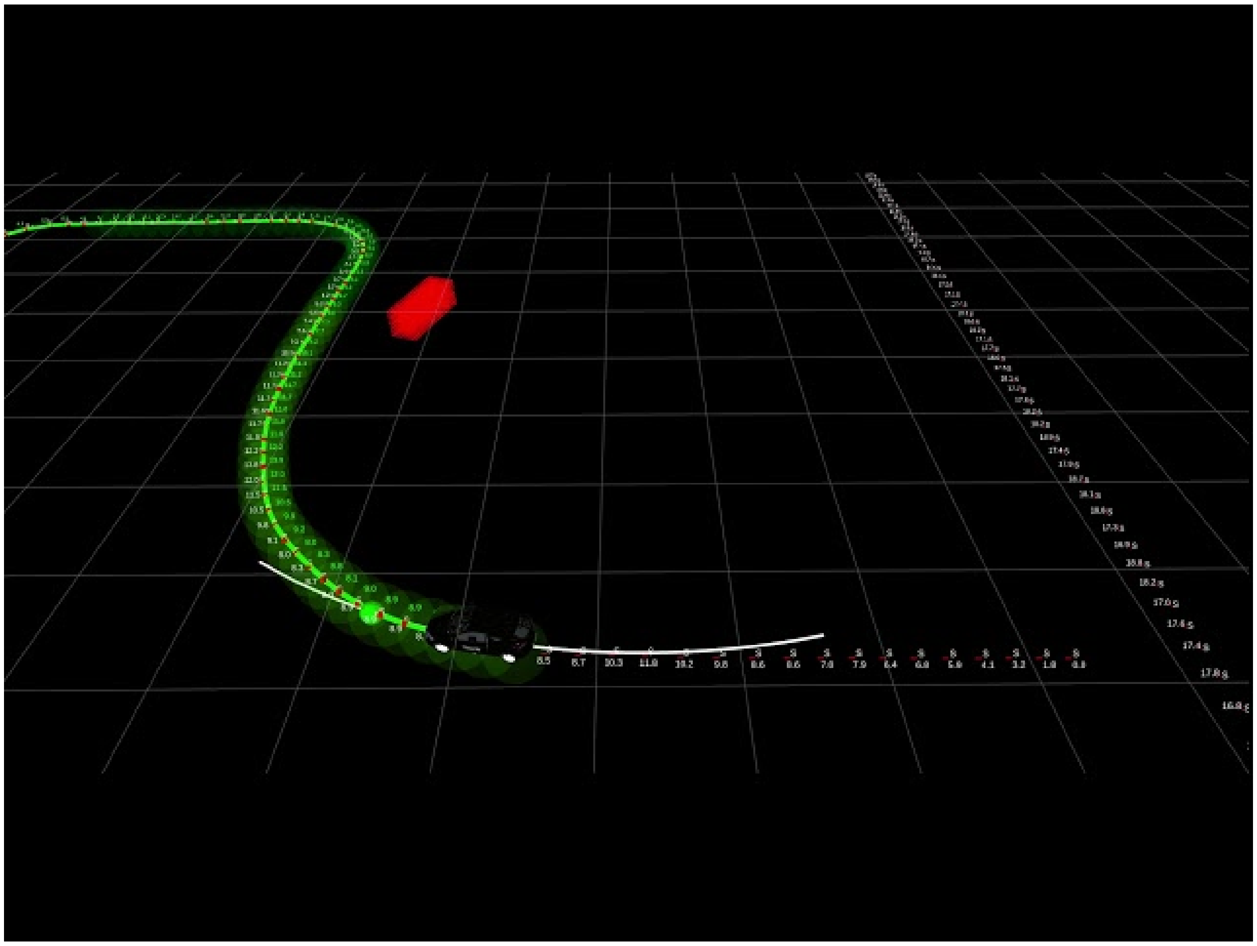
Parameters

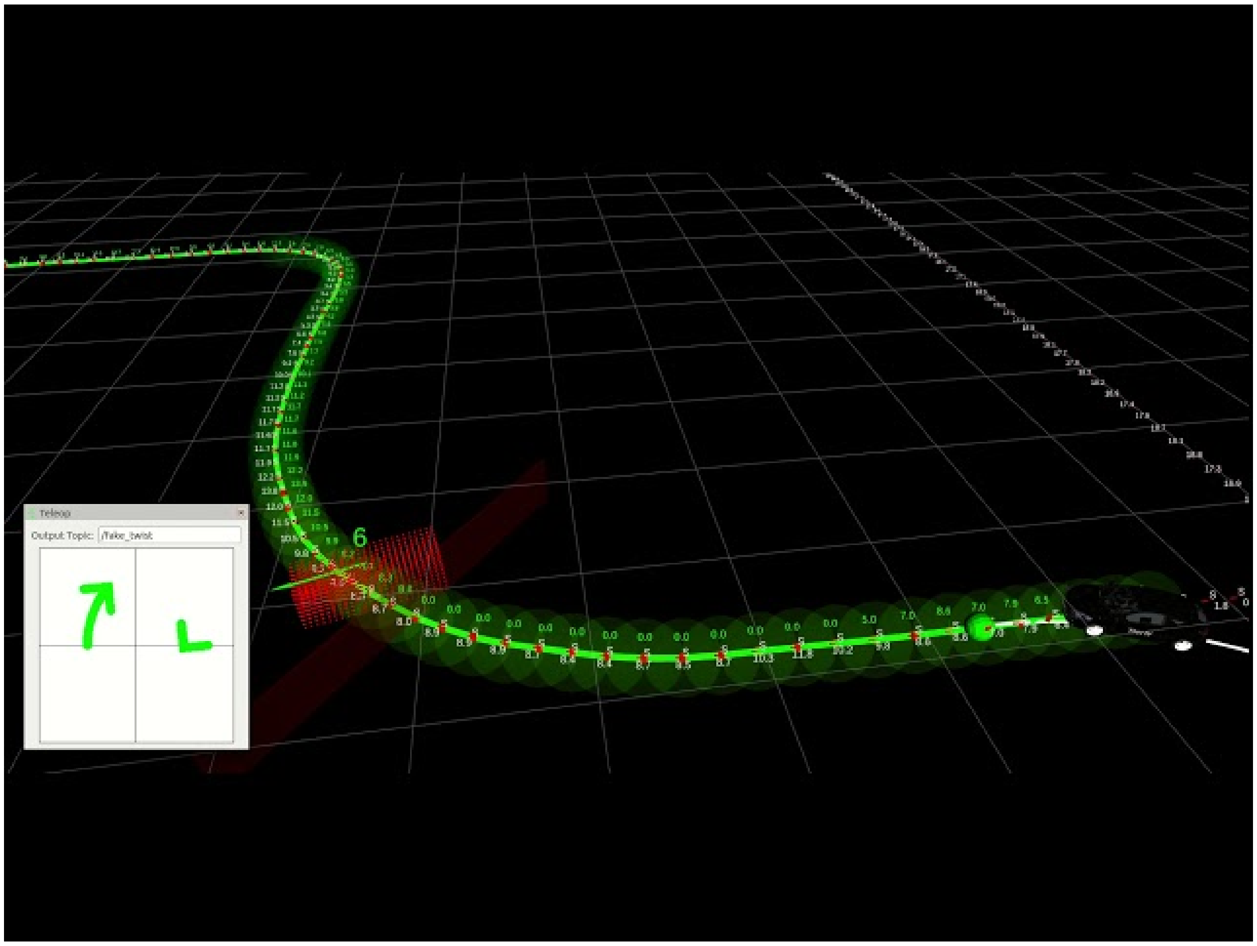
Parameters can be set in both Launch file and Runtime manager:

Subscriptions/Publications

```
Node [/lidar_fake_percetion]
Publications:
 * /fake_objects [autoware_msgs/DetectedObjectArray]
 * /fake_points [sensor_msgs/PointCloud2]

Subscriptions:
 * /move_base_simple/goal [geometry_msgs/PoseStamped]
 * /detected_objects [autoware_msgs/DetectedObjectArray]
 * /points_raw [sensor_msgs/PointCloud2]
 * /fake_twist [geometry_msgs/Twist]
 * /tf [tf2_msgs/TFMessage]
```





IMM-UKF-PDA Tracker

Autoware package based on IMM-UKF-PDA tracker.

- From a sourced terminal:

```
roslaunch lidar_tracker imm_ukf_pda_tracker.launch
```

- From Runtime Manager:

Computing Tab -> Detection/ lidar_detector -> `imm_ukf_pda_tracker`

Reference

A. Arya Senna Abdul Rachman, 3D-LIDAR Multi Object Tracking for Autonomous Driving. 2017. paper

M. Schreire, Bayesian environment representation, prediction, and criticality assessment for driver assistance systems. 2017. paper

Requirements

- `eucledian_cluster` node.
- `ray_ground_filter` node.

- `/tf` topic. Below video is from Suginami data which contais /tf topic: (`autoware-20180205150908.bag`). You can download it from ROSBAG STORE for free. Otherwise, you need to do localization with a map to produce /tf topic from `velodyne` to `world`.
- `wayarea` info from vectormap if is possible.

Parameters

Launch file available parameters for `imm_ukf_pda_tracker`

Launch file available parameters for `visualize_detected_objects`

Subscribed topics

Node: imm_ukf_pda_tracker

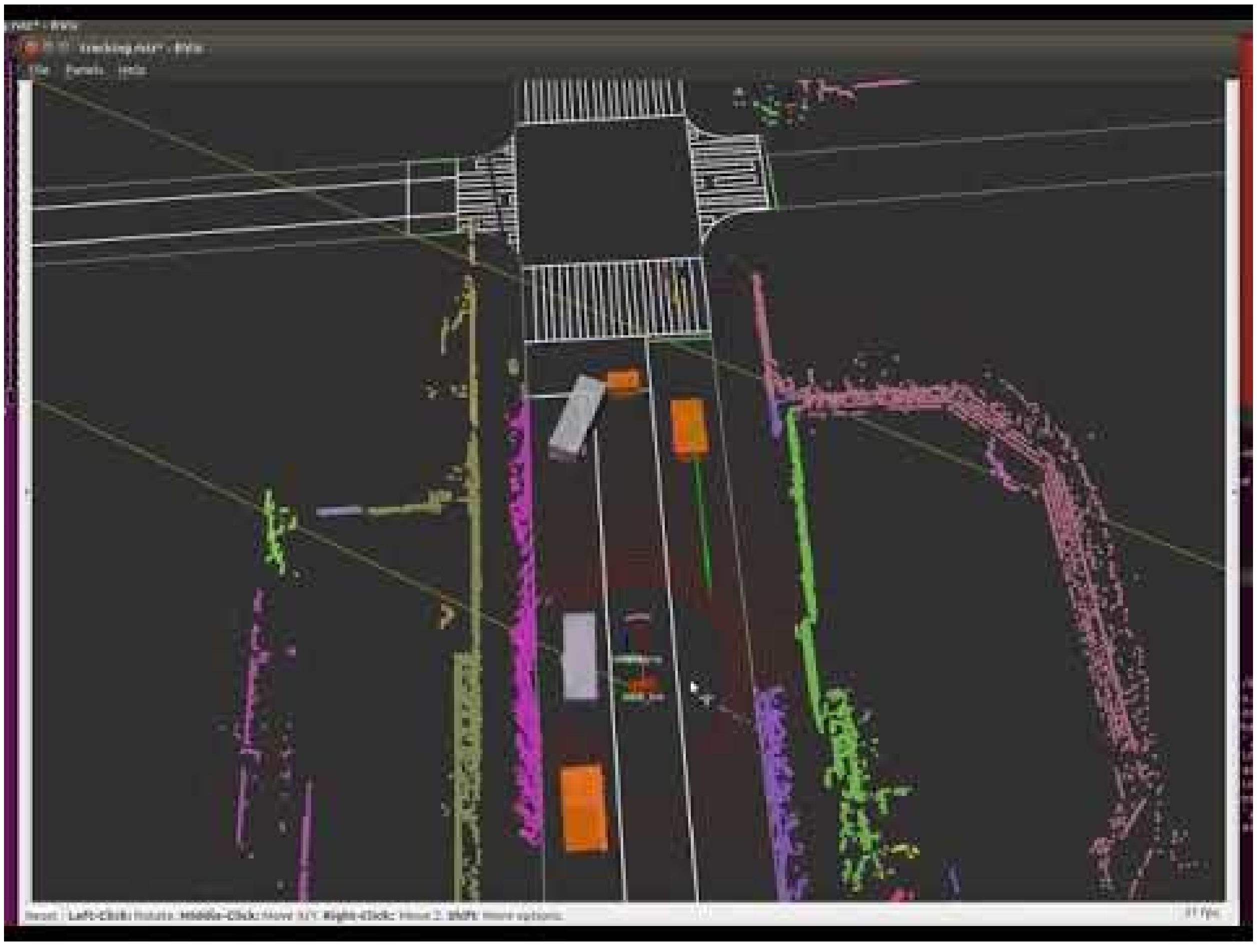
Node: visualize_detected_objects

Published topics

Node: imm_ukf_pda_tracker

Node: visualize_detected_objects

Video



Benchmark

Please notice that benchmark scripts are in another repository. You can tune parameters by using benchmark based on KITTI dataset. The repository is here.

KF based Object Tracker

kf contour tracker

This nodes contains three options of tracking

- Association only
- Simple KF tracking
- Contour area plust memory tracker (divid the horizon into contours and associate memory for each circle, which represent the maximum life time of each object)

Outputs

This tracker output (pose, heading, velocity) in global coordinates.

Object Filtering

It can filter detected objects by size or/and vector map lanes proximity.

Options

It tracks either OpenPlanner simulted vehicles or live detection cluster from “lidar_euclidean_cluster_detect” It can simulated frame by frame testing with fixed time intervals 0.1 second.

Requirements

1. cloud_clusters
2. vector map, in case of using vector map object filtering (to disable using vector map set “vector map filter distance” to 0)
3. op percepion simulator, in case of simulation, also vector map filtering should be disabled.

How to launch

- From a sourced terminal:

```
roslaunch lidar_kf_contour_track lidar_kf_contour_track.launch
```

- From Runtime Manager:

Computing Tab -> Detection -> lidar_tracker -> lidar_kf_contour_tracker

Subscriptions/Publications

```
Publications:
 * /tracked_objects [autoware_msgs::DetectedObjectArray]
 * /detected_polygons [visualization_msgs::MarkerArray]
 * /op_planner_tracked_boxes [jsk_recognition_msgs::BoundingBoxArray]

Subscriptions:
 * /cloud_cluster [autoware_msgs::CloudClusterArray]
 * /current_pose [geometry_msgs::PoseStamped]
 * /vector_map_info/*
```

Demo Movie

Naive-L-Shape fitting

- From a sourced terminal:

```
roslaunch lidar_naive_l_shape_detect lidar_naive_l_shape_detect.launch
```

- From Runtime Manager:

Computing Tab -> Detection/ lidar_detector -> `lidar_naive_l_shape_detect`

Reference

A. Arya Senna Abdul Rachman, 3D-LIDAR Multi Object Tracking for Autonomous Driving. 2017. [paper](https://repository.tudelft.nl/islandora/object/uuid:f536b829-42ae-41d5-968d-13bbaa4ec736

Requirements

- `lidar_eucledian_cluster_detect` node.

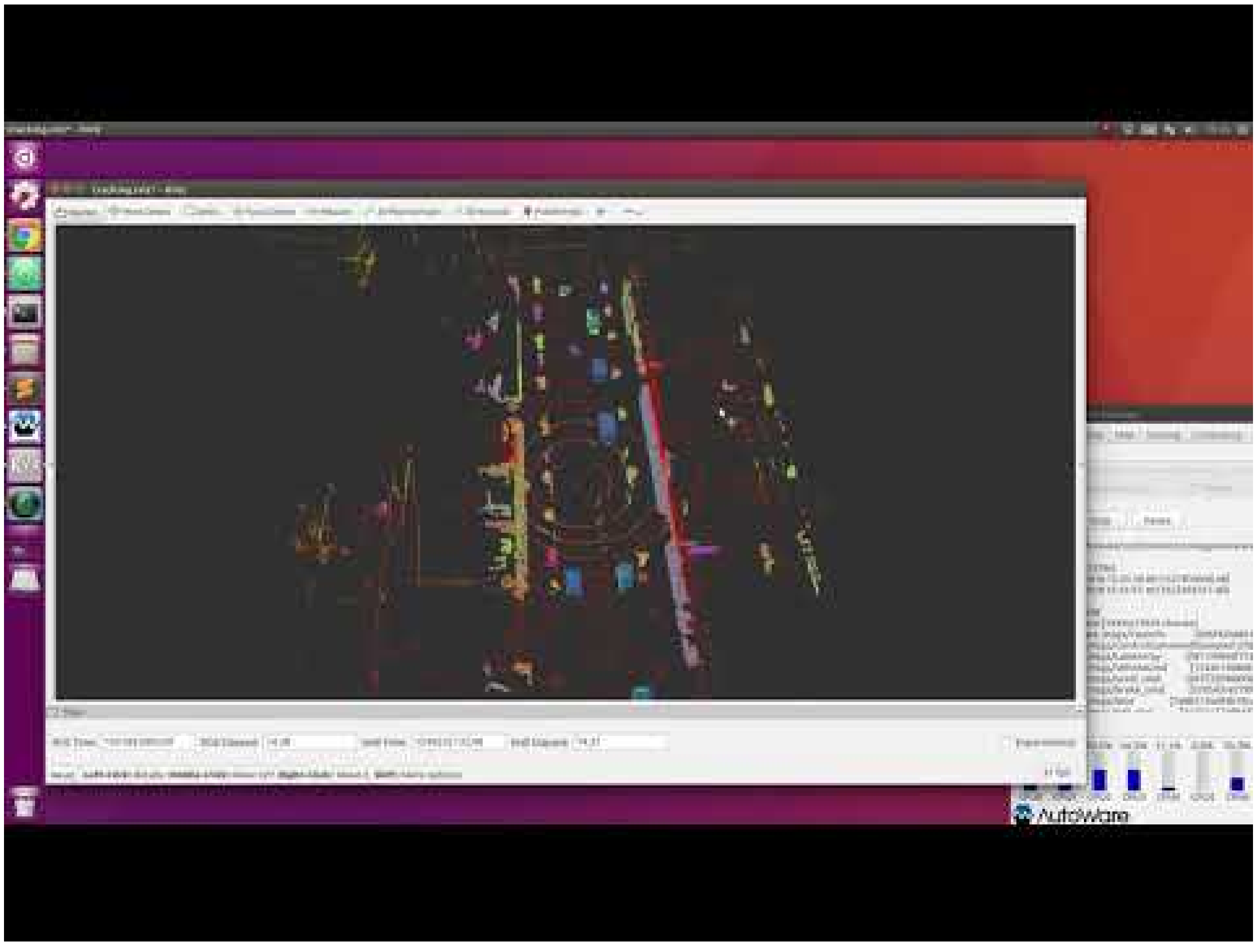
Parameters

Launch file available parameters for `lidar_naive_l_shape_detect`

Subscribed topics

Published topics

Video



LiDAR-Camera Fusion

Pixel-Cloud fusion node

This node projects PointCloud to Image space, extracts RGB information from the Image, back-projects it to LiDAR space, and finally publishes a Colored PointCloud.

Requirements

1. Camera intrinsics
2. Camera-LiDAR extrinsics
3. Image rectified being published

How to launch

- From a sourced terminal:

```
roslaunch pixel_cloud_fusion pixel_cloud_fusion.launch
```

- From Runtime Manager:

Computing Tab -> Fusion -> pixel_cloud_fusion

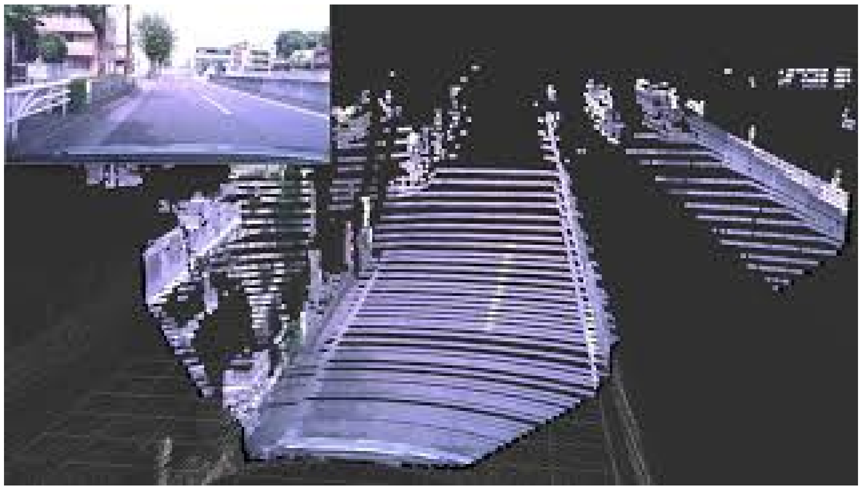
Parameters

Launch file available parameters:

Subscriptions/Publications

```
Node [/pixel_cloud_fusion]
Publications:
 * /points_fused [sensor_msgs/PointCloud2]

Subscriptions:
 * /image_raw [sensor_msgs/Image]
 * /points_raw [sensor_msgs/PointCloud2]
 * /camera_info [sensor_msgs/CameraInfo]
 * /tf [tf2_msgs/TFMessage]
```



ng# Range Vision Fusion

The Range Vision Fusion node will try match the objects detected on a range sensor, with the ones obtained from a vision detector. A match will be considered found if the 3D projection of the object overlaps at least 50% (configurable) over the 2D object detection. The label from the 2D Image detector will be attached to the corresponding 3D Object. All the matched results will be published.

Requirements

Input Topics

1. Camera intrinsics (`sensor_msgs/CameraInfo`)

2. Camera-LiDAR extrinsics (`tf`)
3. Object Detections results from a Vision Detector (`autoware_msgs/DetectedObjectArray`)
4. Object Detections results from a Range Detector (`autoware_msgs/DetectedObjectArray`)

Output Topics

1. Fused Detected Objects (`autoware_msgs/DetectedObjectArray`) on the `/detection/combined_objects` topic.
2. Fused Detected Objects' boxes (`jsk_recognition_msgs/BoundingBoxArray`) on the `/detection/combined_objects_boxes` topic.
3. Fused Detected Objects' labels (`visualization_msgs/MarkerArray`) on the `/detection/combined_objects_labels` topic.

Parameters

Launch file available parameters:

Example of usage

1. Launch a ground filter algorithm from the `Points Preprocessor` in the **Sensing** tab. (adjust the parameters to your vehicle setup).
2. Launch Calibration Publisher with the intrinsic and extrinsic calibration between the camera and the range sensor.
3. Launch a Vision Detector from the Computing tab (this should be publishing by default `/detectoion/vision_objects`).
4. Launch a Lidar Detector from the Computing tab (this should be publishing by default `/detectoion/lidar_objects`).
5. Launch this node.
6. Launch `rviz`, and add the topics shown above in the Output section.

Notes

- Detection on Image space should be performed on a **Rectified** Image, otherwise projection will be incorrect.
- The names of the classes are defined as in the COCO dataset.

Setup Traffic Light recognition based on MxNet

1. Clone the MxNet `v1.0.0` branch from the Apacher MxNet project GitHub onto your Home directory:

```
$ cd
$ git clone -b v1.0.0 --recursive https://github.com/apache/incubator-mxnet.git mxnet
```

1. Install CUDA 8.0 and CUDNN v5.x for CUDA 8.0 as recommended on the GitHub project.
2. Install OpenBLAS

```
$ sudo apt-get install libopenblas-dev
```

1. We recommend to update line 276 from the `Makefile` to only compile the code for your GPU architecture.

```
KNOWN_CUDA_ARCHS := 30 35 50 52 60 61 70
```

i.e. if you own a Pascal card (1060, 1070, 1080, Titan X/Xp)

change it to `KNOWN_CUDA_ARCHS := 61`

For a complete list check https://developer.nvidia.com/cuda-gpus, or execute `deviceQuery` from the CUDA samples to find yours.

1. Compile MxNet with C++, OpenCV, OpenBLAS, CUDA and CUDNN support

```
$ make -j1 USE_BLAS=openblas USE_CUDA=1 USE_CUDA_PATH=/usr/local/cuda
USE_CUDNN=1 USE_CPP_PACKAGE=1
```

1. Compile Autoware
2. Put your trained data in `/tmp`.

   ```
   /tmp
   ├── mxnet-network.json
   └── mxnet-network.params
   ```

   Specify your files name in `Autoware/ros/src/computing/perception/detection/packages/trafficlight_recognizer/launch/traffic_recognition_mxnet.launch`, if you use your own.

For a complete reference on the compilation follow the project's documentation: https://mxnet.incubator.apache.org/get_started/build_from_source.html

Setup region_tlr_ssd

1. Refer this-README to install Caffe for SSD. The compilation process for `region_tlr_ssd` assumes that SSD Caffe has been installed under `~/ssdcaffe/`.Otherwise, specify your Caffe install path in `CMakeLists.txt` for `trafficlight_recognizer` package or the compiler will skip compilation of this node.
2. Compile Autoware
3. Download and decompress trained data(or use your own) under `Autoware/ros/src/computing/perception/detection/packages/rtrafficlight_recognizer/data`.

   Decompression result will be like followings:

   ```
   Autoware/ros/src/computing/perception/detection/packages/trafficlight_recognizer/
   ↪data
   ├── Autoware_tlr_SSD_300x300_iter_60000.caffemodel
   └── deploy.prototxt
   ```

   Specify your caffemodel and prototxt file name in `Autoware/ros/src/computing/perception/detection/packages/trafficlight_recognizer/launch/traffic_recognition_ssd.launch`, if you use your own.
4. That's it!

tl_switch

Introduction

The node uses traffic signal obtained by image recognition and that recieved from external systems, for example AMS, and switch traffic signals safely. If either is red, it turns red. Also, timeout is set for external system information.

How to launch

- From a sourced terminal:

  `roslaunch trafficlight_recognizer traffic_light_recognition.launch`

- From Runtime manager:

  Computing Tab -> region_tlr

Published topics

|Topic|Type|Objective| ——|—-|——— |`/camera_light_color`|`autoware_msgs/traffic_light`|Subscribe traffic light color obtained from camera image.| |`/ams_light_color`|`autoware_msgs/traffic_light`| Subscribe traffic light color recieved from external systems.|

Subscribed topics

|Topic|Type|Objective| ——|—-|——— |`/light_color`|`autoware_msgs/traffic_light`|Publishes traffic light color for planning.|

Beyond Pixels based tracker

This node is based on the work depicted by Beyond Pixels: **Leveraging Geometry and Shape Cues for Online Multi-Object Tracking**. Published on the *Proceedings of the IEEE International Conference on Robotics and Automation*.

How to launch

Example, supposing default topic values.

1. Launch CameraInfo publisher (Sensing Tab -> Calibration Publisher).
2. Launch the Image rectifier (`roslaunch image_processor image_rectifier.launch`).
3. Launch an image based object detector (yolo3, ssd, etc).
4. From a sourced terminal:
   - `roslaunch vision_beyond_track vision_beyond_track.launch`

or from Runtime Manager:

Computing Tab -> Detection/ vision_tracker -> `vision_beyond_track`

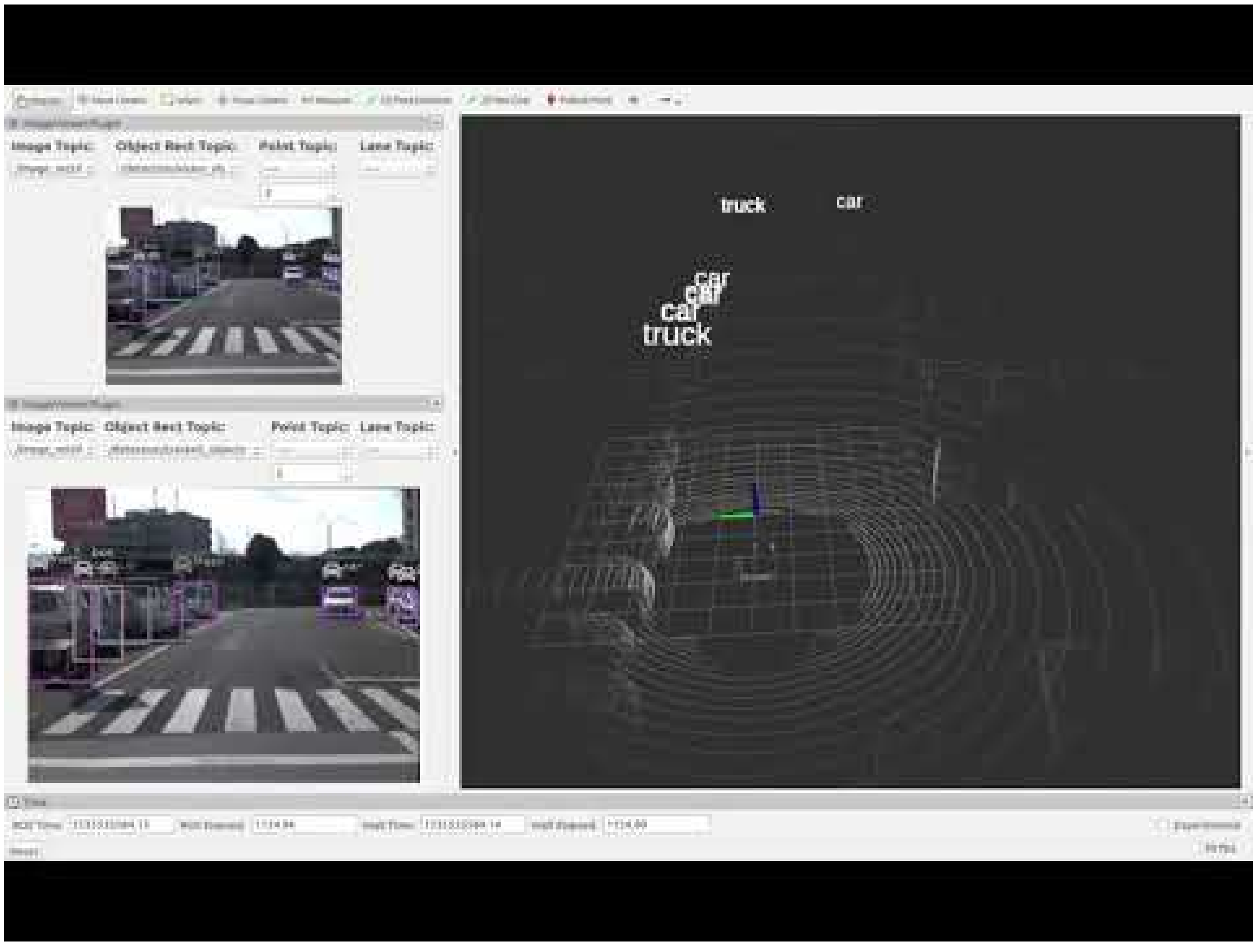
Parameters

Launch file available parameters:

Subscribed topics

Published topics

Video



Vision Darknet Detect

Autoware package based on Darknet that supports Yolov3 and Yolov2 image detector.

Requirements

- NVIDIA GPU with CUDA installed
- Pretrained YOLOv3 or YOLOv2 model on COCO dataset, Models found on the YOLO website.
- The weights file must be placed in `vision_darknet_detect/darknet/data/`.

How to launch

- From a sourced terminal:
  - `roslaunch vision_darknet_detect vision_yolo3_detect.launch`
  - `roslaunch vision_darknet_detect vision_yolo2_detect.launch`
- From Runtime Manager:

Computing Tab -> Detection/ vision_detector -> `vision_darknet_detect` You can change the config and weights file, as well as other parameters, by clicking [app]

Parameters

Launch file available parameters:

Subscribed topics

Published topics

Video

How to Install Caffe for ENet

1. Complete the pre-requisites.
2. Clone ENet:

```
cd ~
git clone --recursive https://github.com/TimoSaemann/ENet.git
cd ENet/caffe-enet
```

1. Compile the ENet fork of Caffe using **Make** *(Don't use CMake to compile Caffe)*. Create `Makefile.config` from `Makefile.config.example` and setup your paths as indicated in http://caffe.berkeleyvision.org/installation.html#compilation.

```
make && make distribute
```

1. Download pre-trained models as provided in https://github.com/TimoSaemann/ENet/tree/master/Tutorial#kick-start, or use your own.

If you didn't install ENet Caffe in `ENet` in your home directory for some reason, modify the Autoware ENet's node `CMakeLists.txt` and point the paths to match your system.

Once compiled, run from the terminal:

```
% roslaunch image_segmenter image_segmenter_enet.launch
```

Remember to modify the launch file located at `computing/perception/detection/packages/image_segmenter/launch/image_segmenter_enet.launch` and configure the network configuration file, the pre-trained models, and the LUT file.

How to Install Caffe for SSD vision detector

Updated as of 2018-07-04

1. Complete the pre-requisites.
2. Clone the SSD Caffe fork in your home directory (CMake files will be looking for it there).

```
% git clone -b ssd https://github.com/weiliu89/caffe.git ssdcaffe
% cd ssdcaffe
% git checkout 4817bf8b4200b35ada8ed0dc378dceaf38c539e4
```

1. Follow the authors' instructions to complete the pre-requisites for compilation.
2. Compile Caffe:

```
make && make distribute
```

1. Download pre-trained models as provided at https://github.com/weiliu89/caffe/tree/ssd, or use your own.

Once compiled, run from the terminal, or launch from RunTimeManager:

```
% roslaunch vision_ssd_detect vision_ssd_detect network_definition_file:=/PATH/TO/
↪deploy.prototxt pretrained_model_file:=/PATH/TO/model.caffemodel
```

Remember to modify the launch file located inside `AUTOWARE_BASEDIR/ros/src/computing/perception/detection/vision_detector/packages/vision_ssd_detect/launch/vision_ssd_detect.launch` and point the network and pre-trained models to your paths.

Launch file params

Notes

Remember to modify the `deploy.prototxt` to use the **FULL PATH** to the VOC label prototxt file.

Open the file `$SSDCAFFEPATH/models/VGGNet/VOC0712/SSD_512x512/deploy.prototxt`

Change line `1803` to point to the full path of the `labelmap_voc.prototxt` file.

2.3.3 Localization

ndt_matching_monitor

Simple health monitoring and reinitialization for `ndt_matching` localization node of Autoware

Introduction

`ndt_matching` publishes the current robot/vehicle position in the `/ndt_pose` topic and statistics in the `/ndt_stat` topic, such as score, execution time.

The score is the result of PCL's function `getFitnessScore()` which measures how well an input point cloud matches the reference pointcloud, in other words the alignment error between the input scan and the reference map. This value can be used to infer the reliability of NDT.

`ndt_matching_monitor` subscribes to `/ndt_stat` and `/ndt_pose` topics, and keeps a running average filter on the score value. When the filtered score is beyond some thresholds, `ndt_matching_monitor` will issue the `/initialpose` topic (same as rviz) using the last known "healty" pose to force `/ndt_matching` to initialize.

If a GNSS device is available, an automatic reinitialization will be triggered with it. `nmea2pose` node is required. Otherwise, a halt in localization will be started. To reset the halted status, use *Initial Pose* tool in RVIZ.

How to launch

- From a sourced terminal:`roslaunch lidar_localizer ndt_matching_monitor.launch`
- From Runtime manager:Computing Tab -> lidar_localizer -> ndt_matching_monitor

Parameters available

Subscribed topics

Published topics

Contributing

`ndt_matching_monitor` is far from complete. For instance, the last known "healthy" pose stated above is just the last `/ndt_pose` value. A Kalman or particle filter might be used to improve this node.

2.3.4 Prediction

Naive Motion Prediction

Autoware package for motion prediction.

- From a sourced terminal:

`roslaunch naive_motion_predict naive_motion_predict.launch`

- From Runtime Manager:

Computing Tab -> Prediction/ Motion Predictor -> `naive_motion_predict`

Requirements

- `ray_ground_filter` node.
- `eucledian_cluster` node.
- `/tf` topic. `velodyne` to `world`.
- `imm_ukf_pda_tracker` node.

Parameters

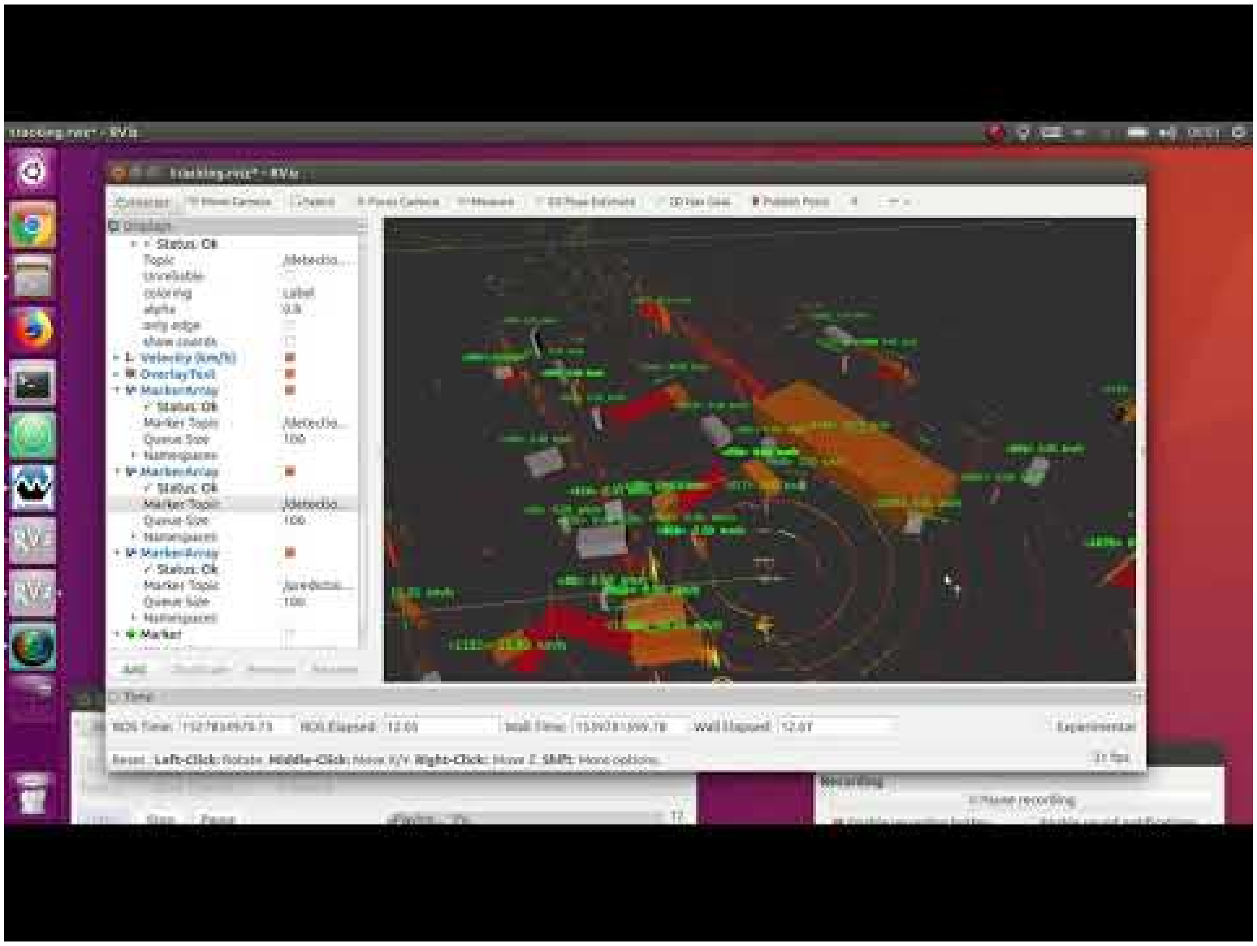
Launch file available parameters for `naive_motion_predict`

|

Subscribed topics

Published topics

Video



2.3.5 Decision

2.3.6 State

2.3.7 Mission Planning

lane_planner

Package

This package has following nodes.

- lane_navi
- lane_rule

- lane_select
- lane_stop

Nodes

lane_select

1. (a)
   (b)
   (c)
   (d)
   (e)
   (f) or
   (g) publish
   (h) publish
2. - ver3waypoint_maker
3. Subscribed Topics
   - traffic_waypoints_array (waypoint_follower/LaneArray)
   - current_pose (geometry_msgs/PoseStamped)
   - current_velocity (geometry_msgs/TwistStamped)
   - state (std_msgs/String)
   - config/lane_select (runtime_manager/ConfigLaneSelect)
4. Published Topics
   - base_waypoints (waypoint_follower/lane)
   - closest_waypoint (std_msgs/Int32)
   - change_flag (std_msgs/Int32)
   - lane_select_marker (visualization_msgs/MarkerArray) : for debug
5. Parameter from Runtime Manager
   - Distance threshold to neighbor lanes
   - Lane Change Interval After Lane Merge
   - Lane Change Target Ratio (m/s) or
   - Lane Change Target Minimum or
   - Vector Length of Hermite Curve

OpenPlanner - Global Planner

op_global_planner

This node generate global path from start point to target point(s) on a map. Planning cost is distance only. the algorithm could be extended to handle other costs such as turning right and left busy lanes in the dynamic map. it supports autoware vector map, and special designed .kml maps.

Outputs

Global path from start to goal, if multiple goals are set, replanning is automatic when the vehicle reaches the end one goal.

Options

Lane change is avilable (parralel lanes are detected automatically) Start/Goal(s) are set from Rviz, and saved to .csv files, if rviz param is disables, start/goal(s) will be loaded from .csv file at.

Requirements

1. vector map

How to launch

- From a sourced terminal:

```
roslaunch op_global_planner op_global_planner.launch
```

- From Runtime Manager:

Computing Tab -> Mission Planning -> OpenPlanner - Global Planner -> op_global_planner

Subscriptions/Publications

```
Publications:
 * /lane_waypoints_array [autoware_msgs::LaneArray]
 * /global_waypoints_rviz [visualization_msgs::MarkerArray]
 * /op_destinations_rviz [visualization_msgs::MarkerArray]
 * /vector_map_center_lines_rviz [visualization_msgs::MarkerArray]

Subscriptions:
 * /initialpose [geometry_msgs::PoseWithCovarianceStamped]
 * /move_base_simple/goal [geometry_msgs::PoseStamped]
 * /current_pose [geometry_msgs::PoseStamped]
 * /current_velocity [geometry_msgs::TwistStamped]
 * /vector_map_info/*
```

2.3.8 Motion Planning

OpenPlanner - Local Planner

previously known as dp_planner, now it is a collection of node with each task of local planning is separated. for more details about OpenPlanner check our paper Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments

to run the local planner user needs to run all nodes in "OpenPlanner - Local planning" for more details about how to run check tutorial video

op_common_params

This node loads the common parameters for the local planner, these parameters are using by op_trajectory_generator, op_motion_predictor, op_trajectory_evaluator, op_behavior_selector and lidar_kf_contour_track.

Outputs

Loads the common parameters for the local planning

Options

- loads params included in the launch file.
- Creates folders for logging and saving important data for all nodes of OpenPlanner folder structure:
- /home/user/autoware_openplanner_logs
  - /BehaviorsLogs
  - /ControlLogs
  - /GlobalPathLogs
  - /PredictionResults
  - /SimulatedCar1
  - /SimulatedCar2
  - /SimulatedCar3
  - /SimulatedCar4
  - /SimulatedCar5
  - /SimulationData
  - /TrackingLogs
  - /TrajectoriesLogs

How to launch

- From a sourced terminal:

```
roslaunch op_local_planner op_common_params.launch
```

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Local planning -> op_common_params

Parameters

- check paper

OpenPlanner - Simulator

Collection of nodes developed to help testing planning algorithm, it could be used with any planner. it consists of three main modules (perception simulator, traffic lights simulator, vehicle simulator)

Demo Movie

op_car_simulator_i

This node simulate a vehicle and its motion, it uses global planning and local planning libraries exactly as OpenPlanner with minor differences. user can launch any number of simulated vehicles with simple customization. currently there are 5 simulated vehicles as an example in autoware.

_i represents the id of the simulated vehicle.

Outputs

simulated vehicle position, TF and its dimentions.

Options

- plan and move from start position to goal position.
- start and goal position are recorder so it starts automatically every time use launches the node.
- user can set auto replay , which start the vehicle again when it arrives to the goal.
- user can log all simulated vehicles internal state
- could be controlled by game wheel manually
- motion could be controled frame by frame for testing with time intervals (0.1) second.

Requirements

1. Vector map
2. Start/Goal points

How to launch

- From a sourced terminal:

```
roslaunch op_simulation_package op_car_simulator_i.launch
```

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Simulator -> op_car_simulator_i

Parameters

- similar to Local planner parameters

op_signs_simulator

This node simulates traffic lights for only one intersection with interchangable traffic lights. user can specify two sets of traffic lights, and the node will switch between them (green, red), yellow is considered as red.

Outputs

- /roi_signal [autoware_msgs::Signals]

Requirements

1. vector map with signal information.

How to launch

- From a sourced terminal:

```
roslaunch op_simulation_package op_signs_simulator.launch
```

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Simulator -> op_signs_simulator

Parameters

- Ids and time for traffic signs first set
- Ids and time for traffic signs second set

op_perception_simulator

This node emulate the object detection using LIDAR data similar to (lidar_euclidean_cluster_detect). The node receives position and dimention from op_car_simulator_i then generate noisy point cloud for each vehicle, then send all data as one cluster_cloud to lidar_kf_contour_track

Outputs

- /cloud_clusters [autoware_msgs::CloudClusterArray]

How to launch

- From a sourced terminal:

```
roslaunch op_simulation_package op_perception_simulator.launch
```

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Simulator -> op_perception_simulator

Parameters

- Maximum number of vehicles that will be simulated

OpenPlanner - Utilities

op_bag_player

OpeGL based rosbag file player, it enable frame by frame playing, but only supports 3 types of topics (lidar data point cloud, current pose, image)

Outputs

publish topics for points_raw, ndt_pose, and image_raw

Options

- play data forward (space)
- pause (space)
- step forward by one frame (up button)
- step backwards by one frame (down button)
- exit (Esc button)

Requirements

1. rosbag path with /pose, /lidar and /image topics recorded

How to launch

- From a sourced terminal:

`roslaunch op_utilities op_bag_player.launch`

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Utilities -> op_bag_player

Parameters

- rosbag file path

Subscriptions/Publications

```
Publications:
 * /points_raw [sensor_msgs::PointCloud2]
 * /ndt_pose [geometry_msgs::PoseStamped]
 * /image_raw [sensor_msgs::Image]

Subscriptions:
```

op_pose2tf

This node recieve ndt_pose and publish current pose TF to the autoware system. so we don't need to run ndt_matching again.

Outputs

pose TF

Requirements

1. ndt_pose topic is published

How to launch

- From a sourced terminal:

`roslaunch op_utilities op_pose2tf.launch`

- From Runtime Manager:

Computing Tab -> Motion Planning -> OpenPlanner - Utilities -> op_pose2tf

Parameters

- ndt_pose topic name

waypoint_maker

Overview

- This package has following nodes.
  - waypoint_clicker
  - waypoint_saver
  - waypoint_marker_publisher
  - waypoint_loader
  - waypoint_velocity_visualizer
- 3-formats of waypoints.csv handled by waypoint_maker
  - ver1 consist of x, y, z, velocityno velocity in the first line

  ex)

  3699.6206,-99426.6719,85.8506 3700.6453,-99426.6562,85.8224,3.1646 3701.7373,-99426.6250,85.8017,4.4036 3702.7729,-99426.6094,85.7969,4.7972 3703.9048,-99426.6094,85.7766,4.7954 3704.9192,-99426.5938,85.7504,4.5168 3705.9497,-99426.6094,85.7181,3.6313 3706.9897,-99426.5859,85.6877,4.0757 3708.0266,-99426.4453,85.6608,4.9097 . . .

  - ver2 consist of x, y, z, yaw, velocityno velocity in the first line

  ex)

  3804.5657,-99443.0156,85.6206,3.1251 3803.5195,-99443.0078,85.6004,3.1258,4.8800 3802.3425,-99442.9766,85.5950,3.1279,7.2200 3801.2092,-99442.9844,85.5920,3.1293,8.8600 3800.1633,-99442.9688,85.5619,3.1308,10.6000 3798.9702,-99442.9609,85.5814,3.1326,10.5200 3796.5706,-99442.9375,85.6056,3.1359,10.2200 3795.3232,-99442.9453,85.6082,3.1357,11.0900 3794.0771,-99442.9375,85.6148,3.1367,11.2300 . . .

  - ver3 category names are on the first line

  ex consist of x,y,z,yaw,velocity,change_flag

  x,y,z,yaw,velocity,change_flag 3742.216,-99411.311,85.728,3.141593,0,0 3741.725,-99411.311,85.728,3.141593,10,0 3740.725,-99411.311,85.733,3.141593,10,0 3739.725,-99411.311,85.723,3.141593,10,0 3738.725,-99411.311,85.719,3.141593,10,0 3737.725,-99411.311,85.695,3.141593,10,0 3736.725,-99411.311,85.667,3.141593,10,0 3735.725,-99411.311,85.654,3.141593,10,0 . . .

Nodes

waypoint_loader

1. Overview
   - Convert waypoints.csv to ROS message type.
   - Correspond to the above 3 types of csv.
   - Adjust waypoints offline (resample and replan velocity)
   - Save waypoints.csv as ver3 format.

2. How to use

- How to start
  - At `Computing`->`waypoint_loader`:
    - Check app->`disable_decision_maker`. If you want to use `decision_maker`, switch to false. Otherwise switch to true.
    - Check `waypoint_loader` and start.
- Idea of velocity replanning
  - The velocity plan is based on the following idea.
    - On a straight line, it accelerates to the maximum speed.
    - At one curve:
      - Finish decelerating before entering the curve. If the curve is sharper, the more deceleration is required.
      - Maintain a constant speed in the curve.
      - Start to accelerate after getting out of the curve.
- Detail of app tab
  - On `multi_lane`, please select multiple input files. If you want lane change with `lane_select`, prepare ver3 type.
  - Check `replanning_mode` if you want to replan velocity.
    - On replanning mode:
      - Check `resample_mode` if you want to resample waypoints. On resample mode, please set `resample_interval`.
      - Velocity replanning parameter
      - `Vmax` is max velocity.
      - `Rth` is radius threshold for extracting curve in waypoints. Increasing this, you can extract curves more sensitively.
      - `Rmin` and `Vmin` are pairs of values used for velocity replanning. Designed velocity plan that minimizes velocity in the assumed sharpest curve. In the *i*-th curve, the minimum radius *ri* and the velocity *vi* are expressed by the following expressions. *vi = Vmax - (Vmax - Vmin)/(Rth - Rmin) * (Rth - ri)*
      - `Accel limit` is acceleration value for limitting velocity.
      - `Decel limit` is deceleration value for limitting velocity.
      - `Velocity Offset` is offset amount preceding the velocity plan.
      - `End Point Offset` is the number of 0 velocity points at the end of waypoints.

1. Subscribed Topics
   - /config/waypoint_loader (autoware_config_msgs/ConfigWaypointLoadre)
   - /config/waypoint_loader_output (std_msgs/Bool)
2. Published Topics
   - /lane_waypoints_array (autoware_msgs/LaneArray)
3. Parameters

- ~disable_decision_maker

waypoint_saver

1. Overview
   - When activated, subscribe `/current_pose`, `/current_velocity`(option) and save waypoint in the file at specified intervals.
   - `change_flag` is basically stored as 0 (straight ahead), so if you want to change the lane, edit by yourself. (1 turn right, 2 turn left)
   - This node corresponds to preservation of ver3 format.
2. How to use

   On app:
   - Ref on the `Save File` and specify the save file name.
   - Check `Save/current_velocity` if you want to save velocity. In otherwise, saved as 0 velocity.
   - Using `Interval`, it is set how many meters to store waypoint.
3. Subscribed Topics
   - /current_pose (geometry_msgs/PoseStamped) : default
   - /current_velocity (geometry_msgs/TwistStamped) : default
4. Published Topics
   - nothing
5. Parameters
   - ~save_filename
   - ~interval
   - ~velocity_topic
   - ~pose_topic
   - ~save_velocity

2.3.9 Actuation

How to build and use AS node in Autoware

1. Clone the autoware repository
2. After cloning autoware get all the submodules.

```
git submodule update --init --recursive
```

1. Build the workspace
2. Ther is no GUI interface to launch this node. Hence use,

```
roslaunch as pacmod_interface.launch
```

2.3.10 Semantics

The `object_map` Package

This package contains nodes to extract information from the LiDAR sensor and from ADAS Maps. The processed result is published as GridMaps (grid_map) and/or occupancy grids(nav_msgs).

Nodes in the Package

points2costmap

This node reads PointCloud data and creates an occupancy grid based on the data contained in it.

The subscribed PointCloud topic should be a processed PointCloud with the ground previously removed. The default input is set as the output PointCloud result from the node `euclidean_clustering` of the `lidar_tracker` package. However, another option might be the output of the `ray_ground_filter` or `ring_ground_filter`, both belonging to the `points_preprocessor` in the *Sensing* package.

The values contained within the OccupancyGrid range from 0-100, representing a probability percentage where 0 represents free space and 100 represents occupied space.

Input topics

Default: `/points_lanes` (PointCloud) from euclidean cluster. It contains filtered points with the ground removed. Another possible option is `/points_no_ground`.

Output topics

`/realtime_cost_map` (nav_msgs::OccupancyGrid) is the output OccupancyGrid, with values ranging from 0-100.

How to launch

It can be launched as follows:

1. Using the Runtime Manager by clicking the `points2costmap` checkbox under the *Semantics* section in the Computing tab.
2. From a sourced terminal executing: `roslaunch object_map points2costmap.launch`.

Parameters available in roslaunch and rosrun

- `resolution` defines the equivalent value of a cell in the grid in meters. Smaller values result in better accuracy at the expense of memory and computing cost (default value: 0.1).
- `cell_width` represents the width in meters of the OccupancyGrid around the origin of the PointCloud (default: 150).
- `cell_height` represents the height in meters of the OccupancyGrid around the origin of the PointCloud (default: 150).

- `offset_x` indicates if the center of the OccupancyGrid will be shifted by this distance, to the left(-) or right(+) (default: 25).
- `offset_y` indicates if the center of the OccupancyGrid will be shifted by this distance, to the back(-) or front(+) (default: 0).
- `offset_z` indicates if the center of the OccupancyGrid will be shifted by this distance, below(-) or above(+) (dDefault: 0).
- `points_topic` is the PointCloud topic source.

---

laserscan2costmap

This node performs the same function as `points2costmap` but uses 2D LaserScan messages to generate the OccupancyGrid.

Input topics

Default: `/scan` (sensor_msgs::LaserScan) from vscan.

Output topics

`/ring_ogm` (nav_msgs::OccupancyGrid) is the output OccupancyGrid with values ranging from 0-100.

How to launch

It can be launched as follows:

1. Using the Runtime Manager by clicking the `laserscan2costmap` checkbox under the *Semantics* section in the Computing tab.
2. From a sourced terminal by executing: `roslaunch object_map laserscan2costmap.launch`.

Parameters available in roslaunch and rosrun

- `resolution` defines the equivalent value of a cell in the grid in meters. Smaller values result in better accuracy at the expense of memory and computing cost (default value: 0.1).
- `cell_width` represents the width in meters of the OccupancyGrid around the origin of the PointCloud (default: 150).
- `cell_height` represents the height in meters of the OccupancyGrid around the origin of the PointCloud (default: 150).
- `offset_x` indicates if the center of the OccupancyGrid will be shifted by this distance, to the left(-) or right(+) (default: 25).
- `offset_y` indicates if the center of the OccupancyGrid will be shifted by this distance, to the back(-) or front(+) (default: 0).
- `offset_z` indicates if the center of the OccupancyGrid will be shifted by this distance, below(-) or above(+) (default: 0).

- `scan_topic` is the PointCloud topic source.

---

wayarea2grid

The ADAS Maps used in Autoware may contain the definition of the road areas, which is useful for computing whether a region is drivable or not. This node reads the data from an ADAS Map (VectorMap) and extracts the 3D positions of the road regions. It projects them into an OccupancyGrid and sets the value to `128` if the area is **road** and `255` if it is not. These values are used because this node uses 8-bit bitmaps (grid_map).

Input topics

`/vector_map` (vector_map_msgs::WayArea) from the VectorMap publisher. `/tf` to obtain the transform between the vector map(grid_frame) and the sensor(sensor_frame) .

Output topics

`/grid_map_wayarea` (grid_map::GridMap) is the resulting GridMap with values ranging from 0-255. `/occupancy_wayarea` (nav_msgs::OccupancyGrid) is the resulting OccupancyGrid with values ranging from 0-255.

How to launch

It can be launched as follows:

1. Using the Runtime Manager by clicking the `wayarea2grid` checkbox under the *Semantics* section in the Computing tab.
2. From a sourced terminal by executing: `roslaunch object_map wayarea2grid.launch`

Parameters available in roslaunch and rosrun

- `sensor_frame` defines the coordinate frame of the vehicle origin (default value: velodyne).
- `grid_frame` defines the coordinate frame of the map (default: map).
- `grid_resolution` defines the equivalent value of a cell in the grid in meters. Smaller values result in better accuracy at the expense of memory and computing cost (default: 0.2).
- `grid_length_x` represents the width in meters of the OccupancyGrid around the origin of the PointCloud (default: 25).
- `grid_length_y` represents the height in meters of the OccupancyGrid around the origin of the PointCloud (default: 0).
- `grid_position_x` indicates if the center of the OccupancyGrid will be shifted by this distance, left(-) or right(+) (default: 0).
- `grid_position_y` indicates if the center of the OccupancyGrid will be shifted by this distance, back(-) or front(+) (default: 0).

---

grid_map_filter

This node can combine sensor, map, and perception data as well as other OccupancyGrids. It generates OccupancyGrids with useful data for navigation purposes. It publishes three different layers. More details are provided in the output topics section below.

Input topics

`/vector_map` (vector_map_msgs::WayArea) from the VectorMap publisher. `/tf` to obtain the transform between the vector map (grid_frame) and the sensor (sensor_frame). `/realtime_cost_map` (nav_msgs::OccupancyGrid) calculated by `points2costmap` in this package.

Output topics

`/filtered_grid_map` (grid_map::GridMap) which contains 3 layers and is also published as in regular OccupancyGrid messages. `distance_transform` (nav_msgs::OccupancyGrid) applies a distance transform to the OccupancyGrid obtained from the `/realtime_cost_map`, allowing us to obtain a gradient of probabilities surrounding the obstacles PointCloud. `dist_wayarea` (nav_msgs::OccupancyGrid) contains a combination of the distance transform and the road data, as described in the `wayarea2grid` node. `circle` (nav_msgs::OccupancyGrid) draws a circle surrounding the `/realtime_cost_map`.

The output topics are configured as described in the `grid_map` package, and the configuration file is inside the `config` folder of the package.

How to launch

It can be launched as follows:

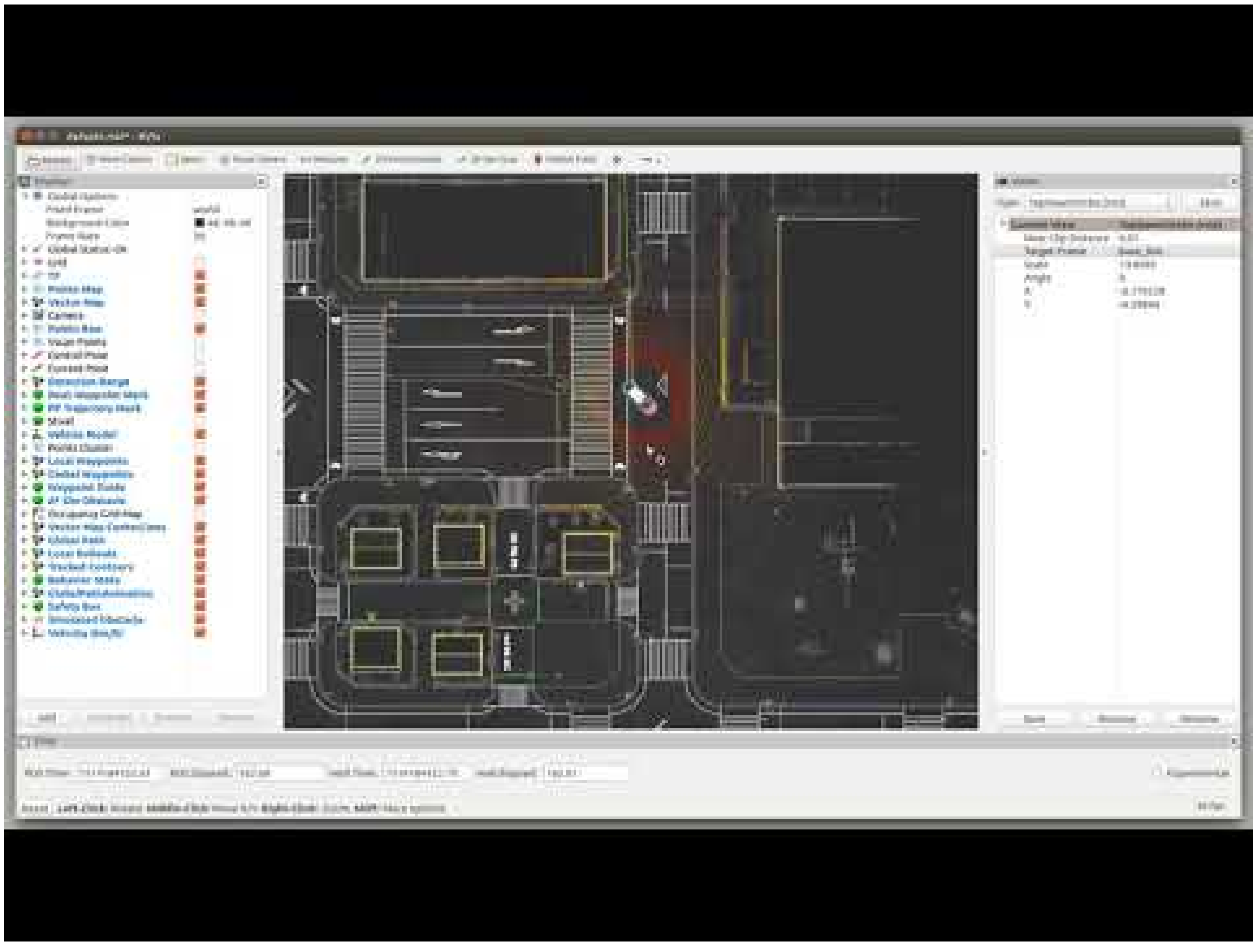
1. Using the Runtime Manager by clicking the `grid_map_filter` checkbox under the *Semantics* section in the Computing tab.
2. From a sourced terminal by executing: `roslaunch object_map grid_map_filter.launch`.

Parameters available in roslaunch and rosrun

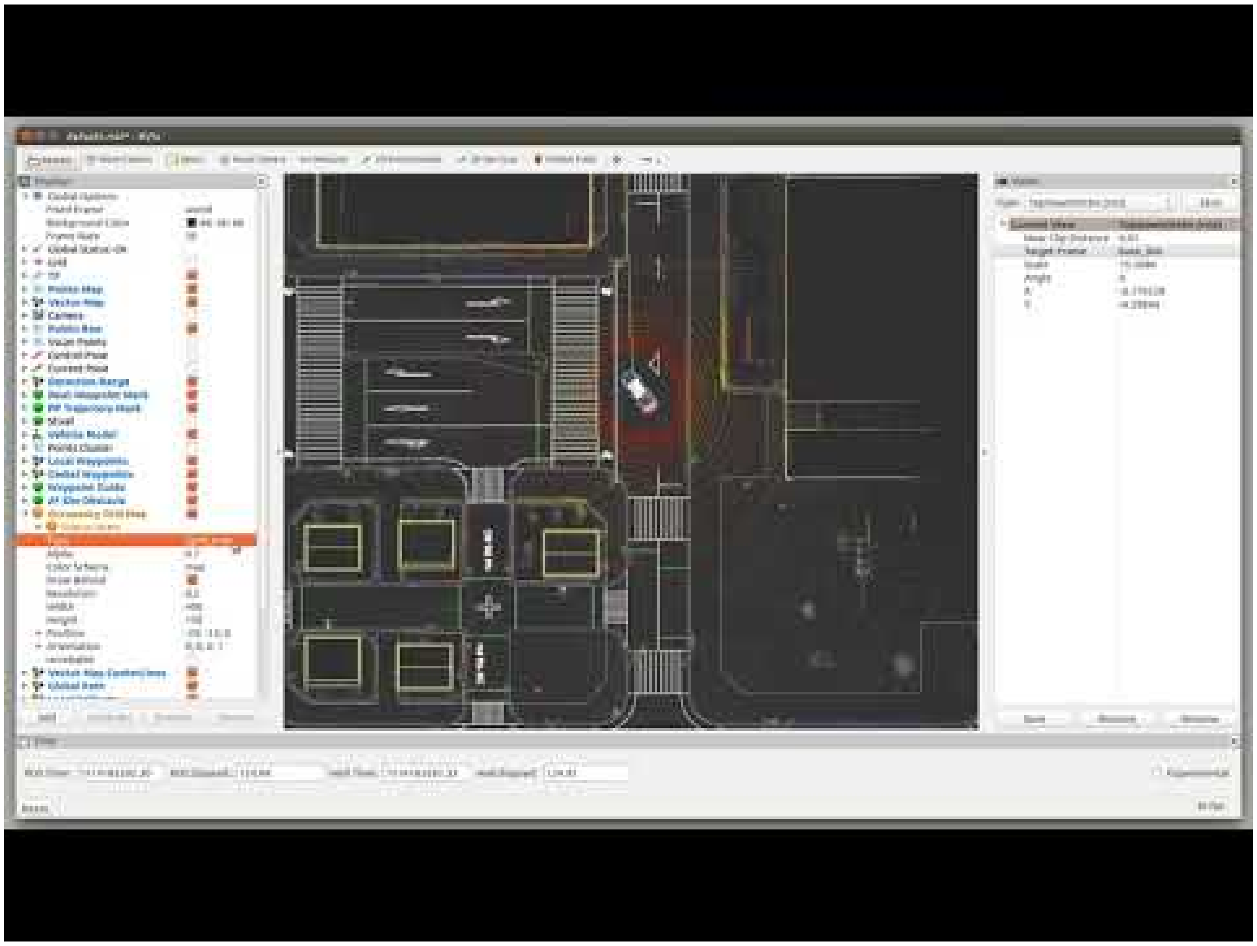
- `map_frame` defines the coordinate system of the realtime costmap (default value: map).
- `map_topic` defines the topic where the realtime costmap is being published (default: /realtime_cost_map).
- `dist_transform_distance` defines the maximum distance to calculate the distance transform, in meters (default: 2.0).
- `use_wayarea` indicates whether or not to use the road regions to filter the cost map (default: true).
- `use_fill_circle` enables or disables the generation of the circle layer (default: true).
- `fill_circle_cost_threshold` indicates the minimum cost value threshold value to decide if a circle will be drawn (default: 20)
- `circle_radius` defines the radius of the circle, in meters (default: 1.7).

Instruction Videos

grid_map_filter



wayarea2grid



Road Occupancy Processor

This package generates the occupancy grid indicating the status of the road. It uses the point cloud from a filtered sensor and the ADAS Map data.

The occupancy grid can be conceptualized as a one dimensional 8-bit depth bitmap.

This package publishes a GridMap/OccupancyGrid with four different possible values:

- UNKNOWN
- FREE
- NO ROAD
- OCCUPIED

These values can be set using parameters, as described in the *Configuration Parameters* section below.

Extra requirements for compilation

- GridMap (http://wiki.ros.org/grid_map)

Prerequisites

- Localization working (TF between velodyne and world)
  - PCD Map (map)
  - TF for the Map (tf map->world)
  - NDT matching (tf world-> base_link)
  - Base link to Localizer (tf base_link -> velodyne)
- Ground filtering method publishing `/points_ground` and `/points_no_ground`. Autoware has three available:
  - `ray_ground_filter`
  - `ring_ground_filter`
  - `euclidean_cluster` with Planar Ground removal
- `wayarea2gridmap` node from Semantics in the Computing Tab
  - Vector Map in `/vector_map` (also known as ADAS Map)
    - VectorMap must contain the definition of road areas (way_areas)

Data subscription

This node subscribes to:

- Ground filtered in `/points_ground` (sensor_msgs::PointCloud)
- Obstacle points in `/points_no_ground` (sensor_msgs::PointCloud)
- GridMap containing the road processed in `/grid_map_wayarea` (grid_map::GridMap)

Data publishing

- `gridmap_road_status` publishes the native format of the grid_map package. This can be used to extend the functionalities of this package.
- `occupancy_road_status` publishes the native ros `nav_msgs::OccupancyGrid` message.

How to run

From a sourced terminal in the workspace: `roslaunch road_occupancy_processor road_occupancy_processor.launch`

Quick Setup

This is only a quick guide, each node must be properly configured

1. Load PointCloud Map (Map tab)
2. Load VectorMap with wayareas available (Map tab)
3. Load TF between map and world frames (Map tab)

4. Load TF between base_link and Localizer (Setup tab)
5. Launch voxel_grid_filter (Sensing tab/Points Downsampler)
6. Launch ray_ground_filter (Sensing tab/Points Preprocessor)
7. Launch ndt_matching (Computing tab/Localization)
8. Initialize NDT pose
9. Launch wayarea2grid (Computing tab/Localization)
10. Launch road_occupancy_processor (Computing tab/Localization)

Configuration parameters

- `points_ground_src` (default=`"points_ground"`) defines the PointCloud source topic containing only the ground points.
- `points_no_ground_src` (default=`"points_no_ground"`) defines the PointCloud source topic containing only the obstacle points.
- `wayarea_topic_src` (default=`grid_map_wayarea`) defines the name of the topic containing the grid_map_msgs:GridMap containing the road areas.
- `wayarea_layer_name` (default=`"wayarea"`) defines the name of the layer in the topic `wayarea_topic_src` that contains the road areas.
- `output_layer_name` (default=`road_status`) defines name of the output layer in the published GridMap object. If running several instances, each vehicle one can publish a different layer and later add them.
- `road_unknown_value` (default=`"128"`) indicates the value to fill in the occupancy grid when a cell is **UNKNOWN**.
- `road_free_value` (default=`"75"`) indicates the value to fill in the occupancy grid when a cell is **FREE**. Should be a number between 0-255.
- `road_occupied_value` (default=`"0"`) indicates the value to fill in the occupancy grid when a cell is **OCCUPIED**. Should be a number between 0-255.
- `no_road_value` (default=`"255"`) indicates the value to fill in the occupancy grid when a cell is **NO ROAD**. Should be a number between 0-255.

Coordinate Frame

The occupancy grid is published in the same coordinate frame as the input GridMap from `/grid_map_wayarea`.

Color representation using default values

Black color represents areas not defined as NO ROAD in the vector map.

Dark gray indicates UNKNOWN areas as detected by this node.

Light gray means the area is FREE.

White is used for OCCUPIED areas.

DevelopersGuide/PackagesAPI/semantics/./doc/road_occupancy_processor.jpg

2.3.11 Util

Modification of a few bag_tools scripts for Autoware.

Original repository: https://github.com/srv/srv_tools bag_tools ROS wiki: http://wiki.ros.org/bag_tools

This repository includes:

- Tool to change frame_id from topics: change_frame_id.py
- Tool to replace timestamp in a message with the header message (unmodified): replace_msg_time_with_hdr.py

change_frame_id

Using this script, you can use change_frame_id.py as before (see bag_tools wiki) and all the specified topics topics will be have their frame_id changed to the same frame_id:

```
rosrun autoware_bag_tools change_frame_id.py -o out.bag -i in.bag -t /camera2/image_
↪raw /camera3/image_raw /camera4/image_raw /camera5/image_raw /camera6/image_raw -f␣
↪camera_frame
```

Alternatively, you specify a 1-to-1 mapping of and topics and frame_id. Below, the frame_id of topic /camera2/image_raw is changed to camera2, the frame_id of the topic /camera3/image_raw is changed to camera3 and so on:

```
rosrun autoware_bag_tools change_frame_id.py -o out.bag -i in.bag -t /camera2/image_
↪raw /camera3/image_raw /camera4/image_raw /camera5/image_raw /camera6/image_raw -f␣
↪camera2 camera3 camera4 camera5 camera6
```

nmea2kml tool

Extract GPS data from rosbag file(s) into .kml and .csv files. .kml file could be viewed by Google Earth. color information indicate the quality of GPS satellite coverage (dark to light red) - (bad - good coverage)

How to Run:

```
rosrun autoware_bag_tools nmea2kml bag_file_name.bag
rosrun autoware_bag_tools nmea2kml bag_files_folder/
```

How to use KITTI Player in Autoware

1. Download any of the KITTI dataset RAW rectified sets and its tracklets (http://www.cvlibs.net/datasets/kitti/raw_data.php). You should have a directory structure that looks like this:

```
kitti_download_dir/2011_09_26/2011_09_26_drive_0001_sync
├── image_00
│   └── data
├── image_01
│   └── data
├── image_02
│   └── data
├── image_03
│   └── data
├── oxts
│   └── data
└── velodyne_points
    └── data
```

1. Create a symlink under `src/util/packages/kitti_pkg/kitti_player` with the name `dataset` that points to your `kitti_download_dir`.

2a. Instead of creating a symlink you can also use the `directory` parameter in the launch file, i.e. `roslaunch kitti_launch kittiplayer.launch directory:=/PATH_TO_KITTI_DATASET/2011_09_26/2011_09_26_drive_0093_sync/`.

1. Modify the `src/util/packages/kitti_pkg/kitti_launch/launch/kittiplayer.launch` file to choose the set you wish to reproduce (lines 5 and 16).
2. Select the frame rate you wish to reproduce by changing the fps argument from the launch file:

```
roslaunch kitti_launch kittiplayer.launch fps:=10
```

In the above example, the player will reproduce the set at 10 fps.

1. The launch file will run `calibration_publisher` and `kitti_box_publisher` nodes and publish:

```
/projection_matrix
/camera/camera_info (sensor_msgs/CameraInfo)
/image_raw (sensor_msgs/Image)
/points_raw (sensor_msgs/PointCloud2)
/kitti_3d_labels (jsk_rviz_plugins/PictogramArray)
/kitti_box (jsk_recognition_msgs/BoundingBoxArray)
/kitti_player/oxts/gps (sensor_msgs/NavSatFix)
/kitti_player/oxts/imu (sensor_msgs/Imu)
```

From version 2, this node aims to play the whole KITTI data into ROS (color/grayscale images, Velodyne scan as PCL, sensor_msgs/Imu Message, GPS as sensor_msgs/NavSatFix Message).

Map Tools

A set of utility tools for map data

PCD Grid Divider

`PCD Grid Divider` creates PCDs divided into grids from PCDs that are not divided into grids.

How to launch

- From a sourced terminal:`rosrun map_tools pcd_grid_divider point_type grid_size output_directory input_pcd1 input_pcd2 ...`

`point_type`: PointXYZ | PointXYZI | PointXYZRGB

`grid_size`: integer (1,5,10,100…)

In the directory you specified, you will see PCDs that divided into grids. The naming rule is `*grid_size*_*lower bound of x*_*lower bound of y*.pcd`

2.3.12 System

About

The catvehicle ROS package houses files that utilize the Gazebo simulator, and additional interfaces to the physical CAT Vehicle Testbed.

Dependencies

- ROS
- obstaclestopper

Catkin Workspace and Build

In order to use the catvehicle ROS package, you should work within a catkin workspace. If you do not already have one:

```
cd ~
mkdir -p catvehicle_ws/src
cd catvehicle_ws/src
catkin_init_workspace
cd ..
catkin_make
```

At this point, you can extract this package into your src directory:

```
cd catvehicle_ws/src
tar xzf catvehicle-x.y.z.tgz
cd ..
catkin_make
```

Simple Tutorial and Examples

Follow the tutorials on the CAT Vehicle Testbed group on the CPS Virtual Organization to see how to use the testbed.

Acknowledgements

License

Authors and contributors

- Jonathan Sprinkle (sprinkjm@email.arizona.edu)
- Rahul Bhadani (rahulkumarbhadani@email.arizona.edu)
- Sam Taylor
- Kennon McKeever (kennondmckeever@email.arizona.edu)
- Alex Warren
- Swati Munjal (smunjal@email.arizona.edu)
- Ashley Kang (askang@email.arizona.edu)
- Matt Bunting (mosfet@email.arizona.edu)
- Sean Whitsitt

Support

This work was supported by the National Science Foundation and AFOSR under awards 1521617, 1446435, 1262960 and 1253334. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

2.3.13 Common

diagnostic_lib

diagnostic packages for Autoware.

packages

1. diag_libpackages contain diag_manager.

2. diag_msgsoriginal messages for diagnostic
3. fake_autoware_nodestest fake nodes and samples

How to test

```
roslaunch fake_autoware_nodes diag_sample.launch
```

How to use

1. add diag_lib,diag_msgs to your package dependency.
2. #include <diag_lib/diag_manager.h> to your source code.
3. create diag_manager class instance in your source code.
4. define error code to yaml file (sample: <diag_lib>/config/error_code_sample_config.yaml)

Compile options

1. -DSTRICT_ERROR_CODE_CHECK=(ON or OFF)if you set this option ON, check error code infomation and when it was wrong, error message was shown and the wrong node was killed.

How it works

diag_manager instance which attatched to your C++ ROS node publish <your_node_name>/diag topic (type:diag_msgs/diag_error).You can define errors in .yaml setting file.Each diag_manager and watchdog node read .yaml setting file and load error codes.The watchdog node check allive or dead all the nodes in the .yaml setting file and aggregate /diag topic which comes from diag_managers.

DevelopersGuide/PackagesAPI/common/rosgraph.png

How to attach diag_manager class

Please read source codes in fake_autoware_nodes

required parameters(rosparam)

error_code_config_path (type:String) : full path of error code config file path.

public member functions

constructer and destructer

1. constructer

```
diag_manager::diag_manager();
```

1. destructor

```
diag_manager::diag_manager();
```

diag functions

Note : argument num is the error number you defined in the error code setting yaml file.

1. DIAG_ASSERT_VALUE_RANGE

```
template<typename T>
void DIAG_ASSERT_VALUE_RANGE(T min, T max, T value, int num)
```

If value is not in the range, send diag message.When you use this function , you expected the value satisfy the conditional expression "min < value < max".num is the error number you defined in the error code setting yaml file.

1. DIAG_ASSERT_VALUE_MIN

```
template<typename T>
void DIAG_ASSERT_VALUE_MIN(T min, T value, int num)
```

If value is not in the range, send diag message.When you use this function , you expected the value satisfy the conditional expression "min < value".num is the error number you defined in the error code setting yaml file.

1. DIAG_ASSERT_VALUE_MAX

```
template<typename T>
void DIAG_ASSERT_VALUE_MIN(T max, T value, int num)
```

If value is not in the range, send diag message.When you use this function , you expected the value satisfy the conditional expression "max < value".num is the error number you defined in the error code setting yaml file.

1. DIAG_ASSERT_EXCEPTION

```
template<class T>
void DIAG_ASSERT_EXCEPTION(T exception,int num)
```

You shold call this function in catch block in your source code.When this function was called, diag_manager send a EXCEPTION diag message.

1. DIAG_RESOURCE

```
void DIAG_RESOURCE(std::string target_resource_path, int num);
```

When the function was called, diag_manager check the target resource file exist.If diag_manager failed to find the file, diag_manager kill the target ROS node.I strongly recommend that you should call this function in the initialize function.num is the error number you defined in the error code setting yaml file.

1. DIAG_RATE_CHECK

```
void DIAG_RATE_CHECK(int num);
```

Put this function where you want to know operation cycle.num is the error number you defined in the error code setting yaml file.

1. DIAG_LOW_RELIABILITY

```
void DIAG_LOW_RELIABILITY(int num);
```

You call this function when your node output is low reliability.num is the error number you defined in the error code setting yaml file.

logging function

1. WRITE_LOG Write log file to /tmp/Autoware/Diag/Log//log.txtThe log remains in such format.

```
[2018-09-13T03:25:25.340543] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:25.341312] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.441295] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.541326] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.641427] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.741318] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.841311] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:25.941436] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:26.041322] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:26.141353] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:26.340464] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:27.340491] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:28.241331] : in /ndt_matching: exception was catched
[2018-09-13T03:25:28.241375] : in /ndt_matching: Divided by zero.
[2018-09-13T03:25:28.340549] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:29.340556] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:30.340551] : in /ndt_matching: topic /nft_matching/data subscribe␣
↪rate was low (Warn)
[2018-09-13T03:25:30.341377] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:30.441393] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:30.541348] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
[2018-09-13T03:25:30.641382] : in /ndt_matching: The input value hogehoge is out of␣
↪range.
```

(continues on next page)

(continued from previous page)

```
```

CHAPTER 3

Community

CHAPTER 4

ToDo List

Todo: Insert document

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/autoware/checkouts/feature-documentation_rtd/docs/UsersGuide/Installation/DockerImage/NVIDIA_DRIVE.rst, line 53.)

Todo: Delete link to GitHub and Import README.md in vision_darknet_detect and vision_ssd_detect.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/autoware/checkouts/feature-documentation_rtd/docs/UsersGuide/Installation/SetupDNN.rst, line 10.)

Todo: Insert link to ROS install

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/autoware/checkouts/feature-documentation_rtd/docs/UsersGuide/Installation/SourceBuild.rst, line 17.)

Todo: Insert link to CUDA install

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/autoware/checkouts/feature-documentation_rtd/docs/UsersGuide/Installation/SourceBuild.rst, line 24.)

Todo: Insert a figure.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/autoware/checkouts/feature-documentation_rtd/docs/UsersGuide/Tutorials/HowToStart.rst, line 14.)

CHAPTER 5

Indices and tables

- genindex
- search