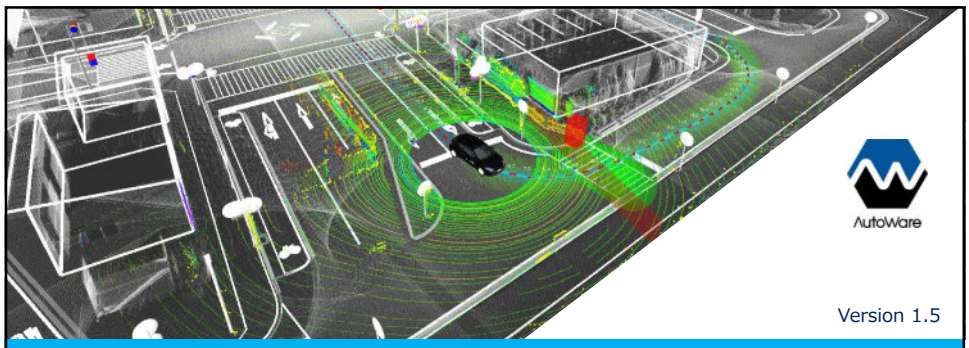


Version 1.5

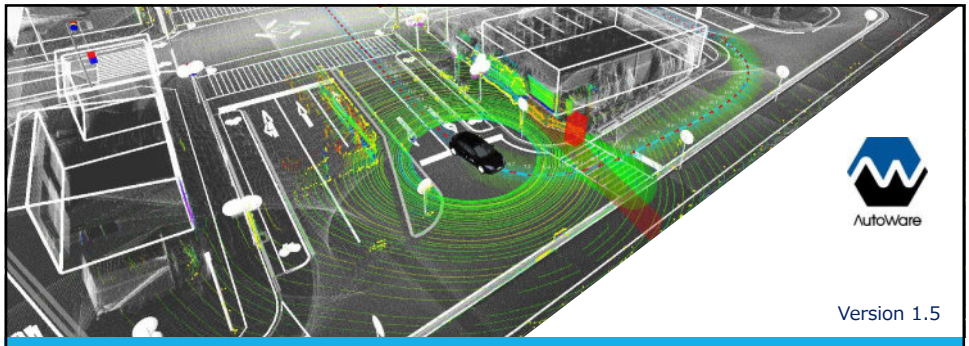
# Autoware in depth

**Day 3**

## Autoware Hands-on Experience

Tier IV
Intelligent Vehicle

## Table of Contents



Version 1.5

Autoware Hands-on Experience

# Chapter 4: Object Detection and Tracking

## 1. Image based detection

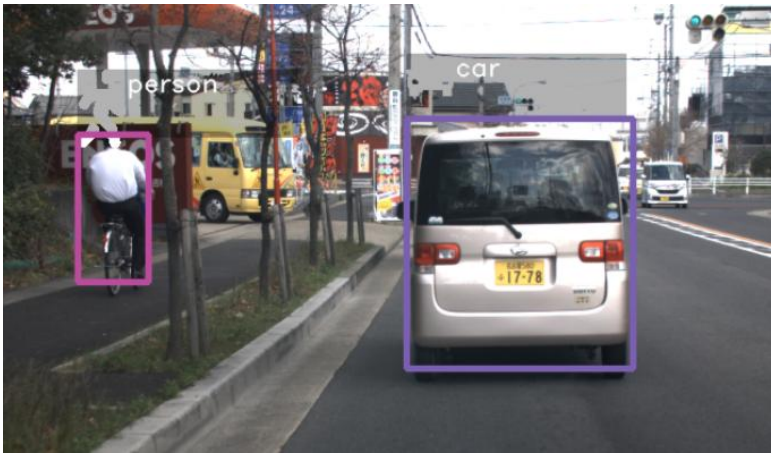
## Object Detection

We need to detect all object not included in our map:

- Any static objects
- Dynamic objects (cars, pedestrians, cyclists)

## Image Based Object and Traffic Light Detection

- Need to detect dynamic object in the scene, **cars**, **pedestrians, cyclists**…

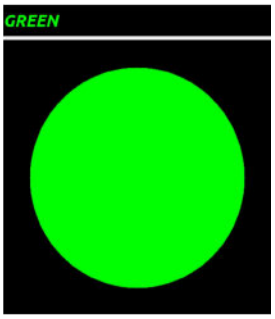



Tier IV Intelligent Vehicle

5

## Image Based Object and Traffic Light Detection

- Need to be able to locate information relevant to planning like **traffic lights**, or unanticipated road conditions.


Tier IV Intelligent Vehicle

6

## Object Detection using Machine Learning

Supervised machine learning really shines for object detection


Tier IV Intelligent Vehicle

7

## Object Detection with YOLOv3

**YOLO – You Only Look Once**

- A **real-time** object detection framework, **not only for driving**
- Performs a single neural network evaluation per image frame
  - Much faster than recurrent methods like R-CNN
- Different models (ex. YOLOv3 and YOLOv3-tiny) for different hardware.


Tier IV Intelligent Vehicle

8

## YOLOv3 in Autoware


**Before starting**

- Model weights **are not** included in Autoware Github
- Download/copy the following files:

```
yolov3.weights
yolov3-tiny.weights
```

and copy to the following location (create if necessary):

```
ros/src/computing/perception/detection/vision_detector/packages/vision_darknet_detect/darknet/data/
```



## YOLOv3 Object Detection – Messages

Autoware uses one msg type at all steps of perception pipeline,
Vision Darknet Detect publishes: **/detection/vision_objects**

DetectedObjectsArray.msg

```
std_msgs/Header          header
DetectedObjects[]        objects
```

DetectedObjects.msg

```
std_msgs/Header                 header
uint32                  id
string                          label
float32                 score   // Score as defined by the detection, Optional
std_msgs/ColorRGBA              color     // Define this object specific color

# 3D information
string                  space_frame // 3D coordinate frame of the object, required if pose and dimensions are defined
geometry_msgs/Pose              pose
geometry_msgs/Vector3           dimensions
geometry_msgs/Vector3           variance
geometry_msgs/Twist             velocity
geometry_msgs/Twist             acceleration
sensor_msgs/PointCloud2         pointcloud
geometry_msgs/PolygonStamped    convex_hull
autoware_msgs/LaneArray         candidate_trajectories
bool                            pose_reliable
bool                            velocity_reliable
bool                            acceleration_reliable

# 2D information
string                          image_frame     // Image coordinate Frame,       Required if x,y,w,h defined
int32                            x          // X coord in image space(pixel) of the initial point of the Rect
int32                           y          // Y coord in image space(pixel) of the initial point of the Rect
int32                           width              // Width of the Rect in pixels
int32                           height             // Height of the Rect in pixels
float32                 angle              // Angle [0 to 2*PI), allow rotated rects

sensor_msgs/Image               roi_image
```

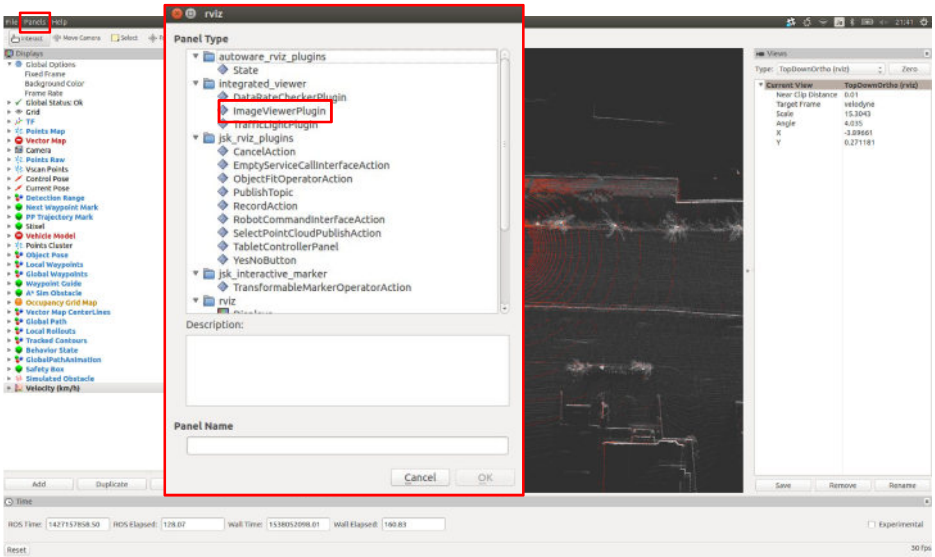
## YOLOv3 Object Detection – Setup and Start

Configure and start the YOLOV3 module from the [Computing] tab.




## YOLOv3 Object Detection – Visualization Step 2

On RViz taskbar click Panels -> Add New Panel -> ImageViewerPlugin



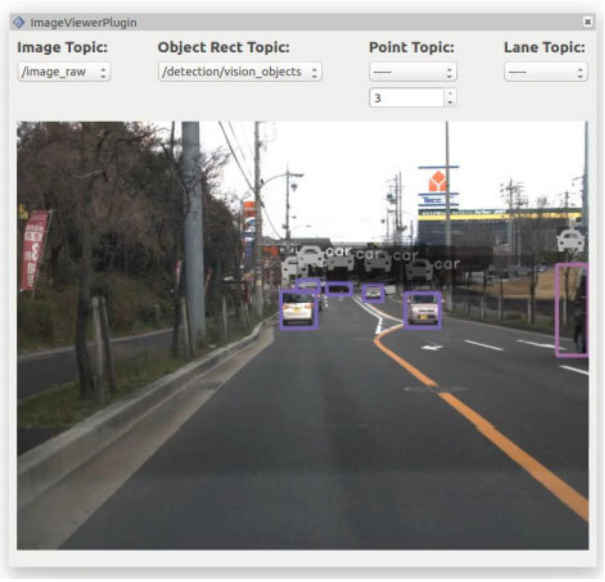
## YOLOv3 Object Detection – Visualization Step 3

Select Image Topic: /image_raw and Object Rect Topic: /detection/vision_objects


Tier IV Intelligent Vehicle



## YOLOv3 Object Detection – Visualization Step 3




Tier IV Intelligent Vehicle

## Tracking Objects

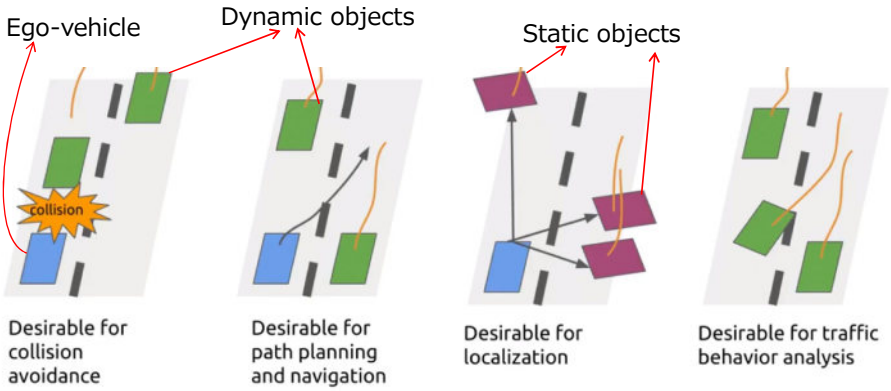
YOLO is frame-to-frame, but we'd really like to keep track of cars


Tier IV Intelligent Vehicle

15

## Object Tracking

- **Tracking** is essential for many tasks related to autonomous driving.
- **Goal:** know which object we've seen before

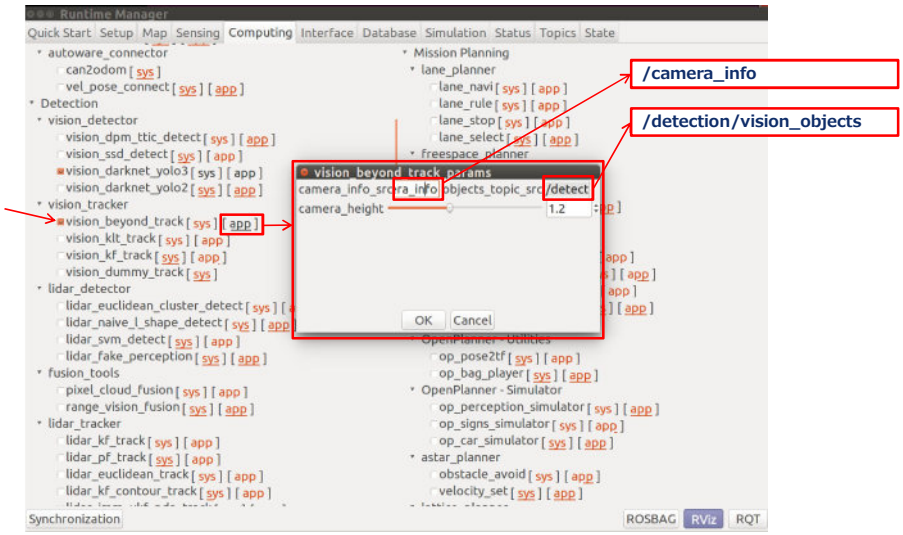



Tier IV Intelligent Vehicle

16

## Tracking Objects Step 2 – Setup Beyond Pixel Tracker

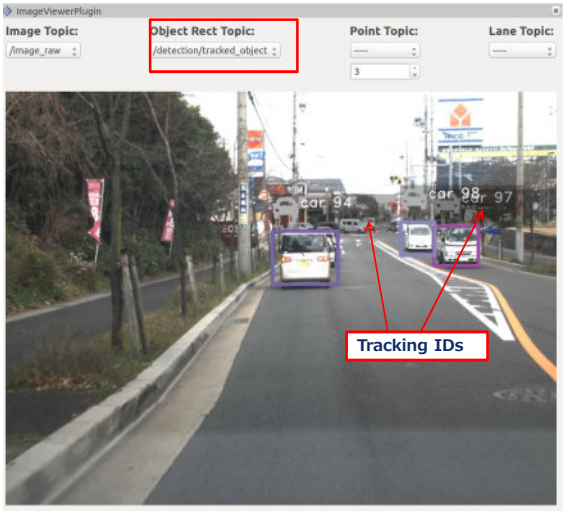
Go back to the [Computing] tab, select [vision_beyond_track].



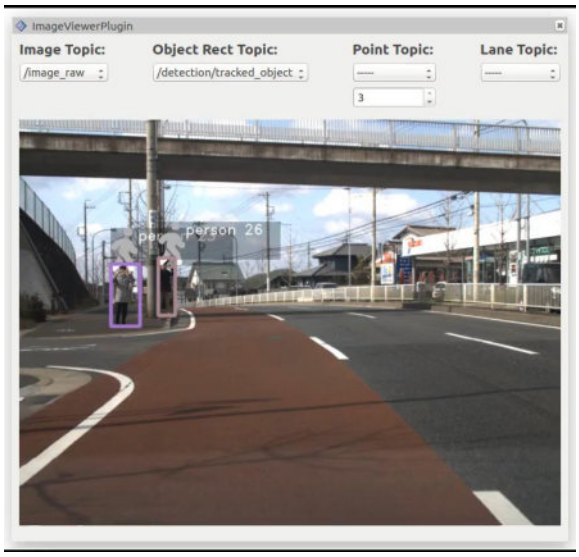



## Tracking Objects Step 3 – Visualization

In the RViz ImageViewerPlugin change the [Object Rect Topic]:

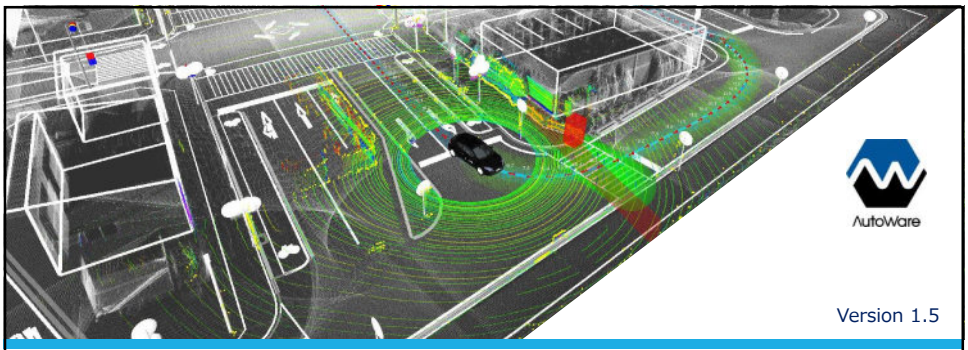


## Tracking Objects Step 3 – Visualization




Tier IV Intelligent Vehicle





AutoWare

Version 1.5

Autoware Hands-on Experience

## Chapter 4: Object Detection and Tracking

2. Traffic Light Detection

Tier IV Intelligent Vehicle

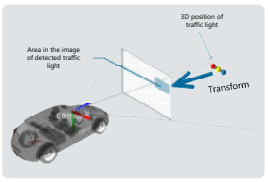
## Traffic Light Detection

- Detects traffic light color from camera images
- Calculates the coordinates of traffic lights from the current position and VectorMap information obtained by localization
- Starting/Stopping at traffic light can be enabled by linking the detection result with a path planning node




## Traffic Light Detection – Structure

※ Some topics are omitted




/vector_map_info

/tf

/feature_projection

Extracts traffic light positions from TF between the Vector Map the camera-LIDAR

/roi_signal

/traffic_light_recognition

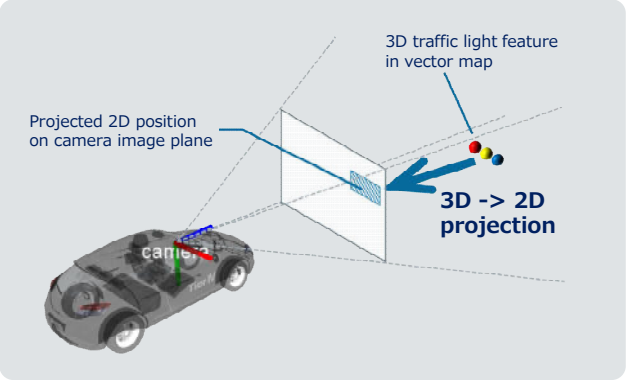
Recognizes traffic the light colors from the ROI values

/light_color

## Feature Projection From Vector Map

Traffic light features from the vector map must be projected into the camera frame.

- Current position comes from NDT matching (relative to LiDAR)
- Extrinsics between LiDAR and camera must be known

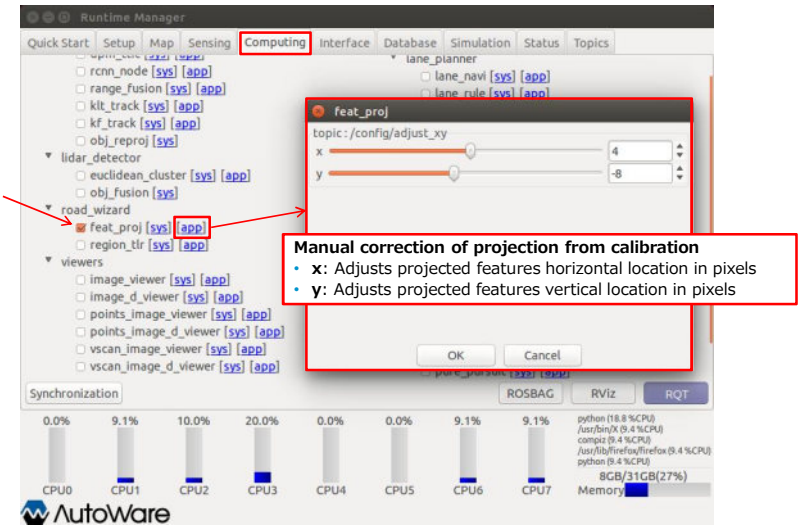



Tier IV

23

## Traffic Light Detection Step 2 – Feature Projection

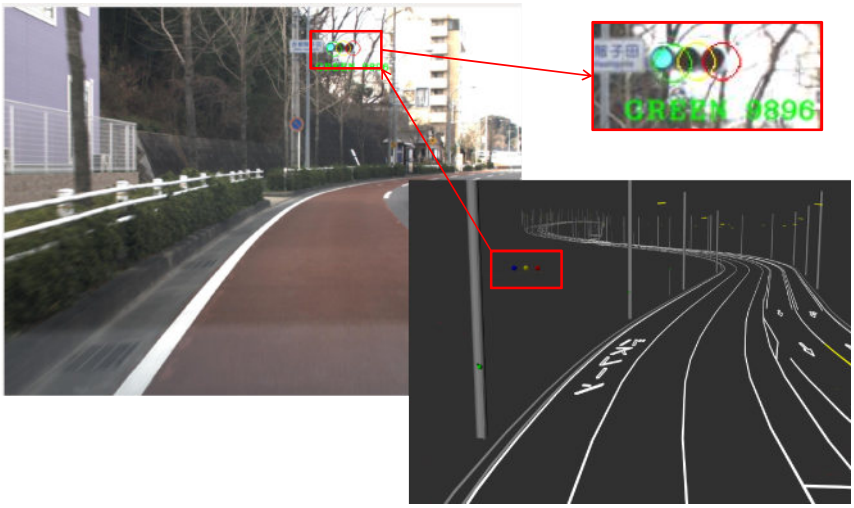
Configure and select 'feat_proj' in the Computing tab




Tier IV

24

## Traffic Light Detection Step 2 – Feature Projection

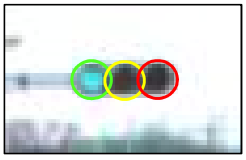



Tier IV

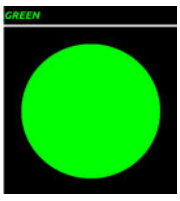
25

## Traffic Light Classification


Traffic light image patch from feature projection



Color thresholding



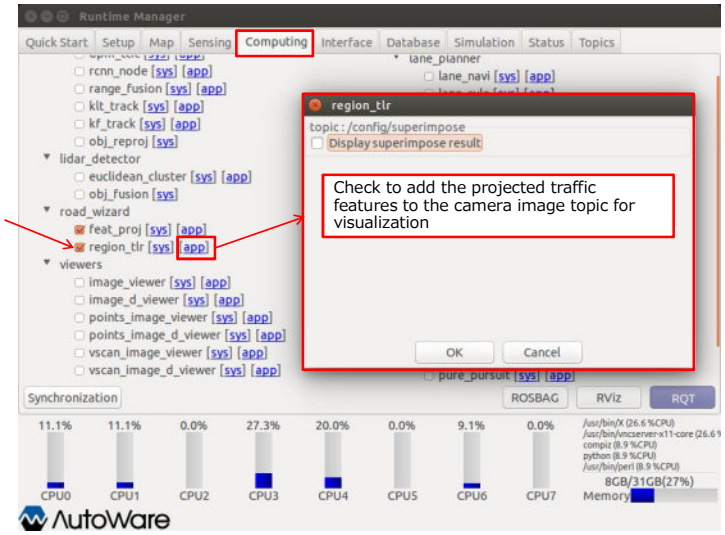

Detection result

Tier IV

26

## Traffic Light Detection Step 3 – Classification

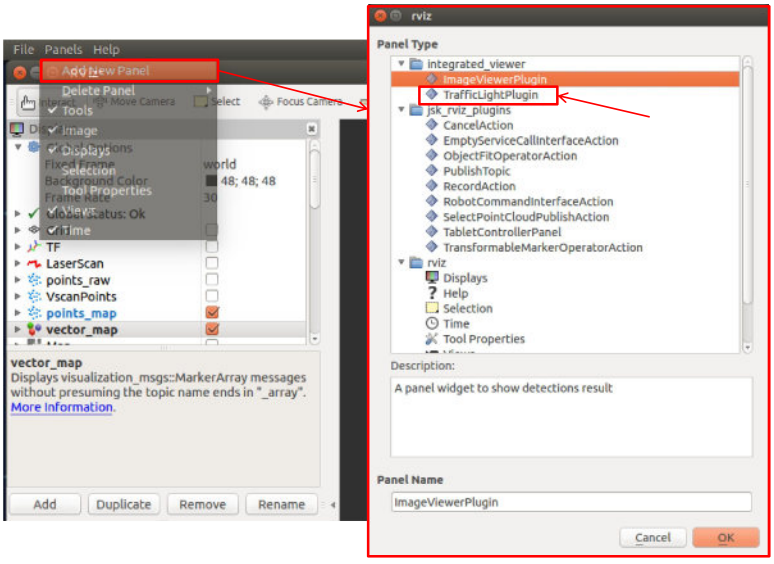
Configure and select [region_tlr] in the [Computing] tab





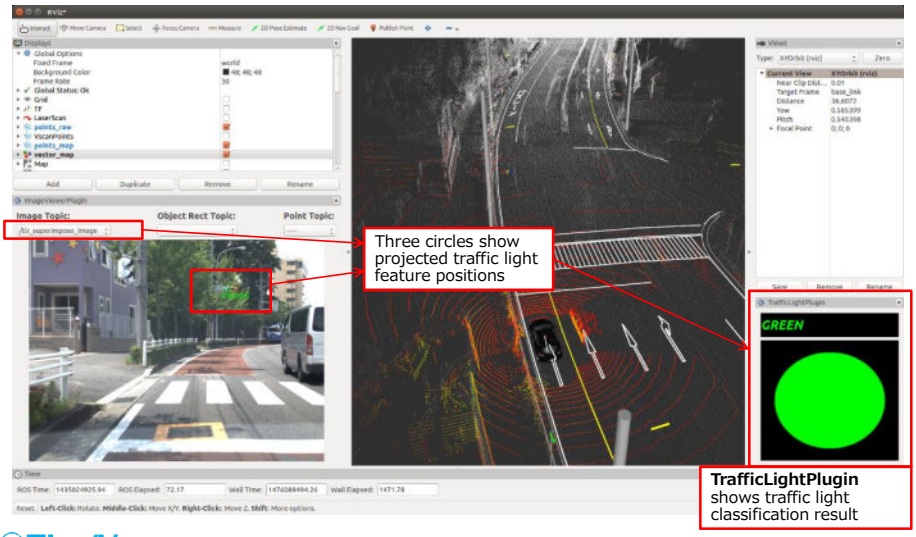

## Traffic Light Detection Step 4 – Visualization

On RViz taskbar click Panels -> Add New Panel -> TrafficLightPlugin

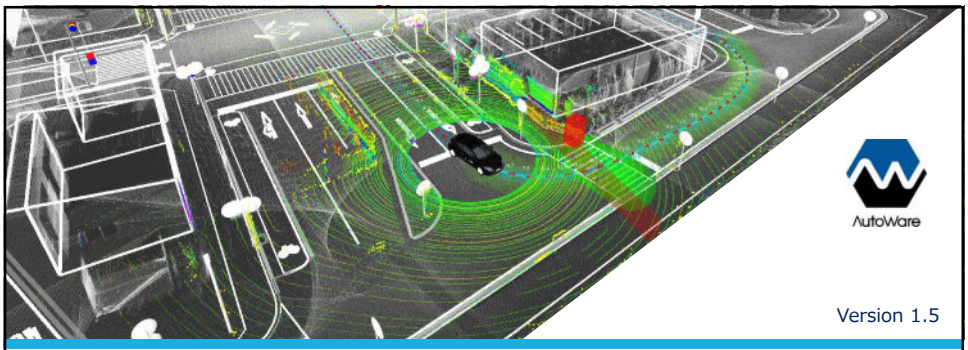


## Traffic Light Detection Step 4 – Visualization

Change [Image Topic:] on [ImageViewerPlugin] to [/tlr_superimpose_image]





Version 1.5

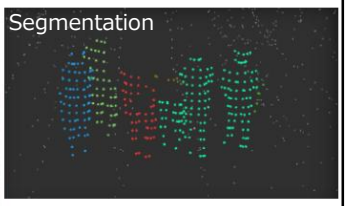
Autoware Hands-on Experience

## Chapter 4: Object Detection and Tracking

3. LiDAR based detection and tracking

## LiDAR Object Detection

Goal: **Segment, cluster, track, and classify** *LiDAR point cloud data.*





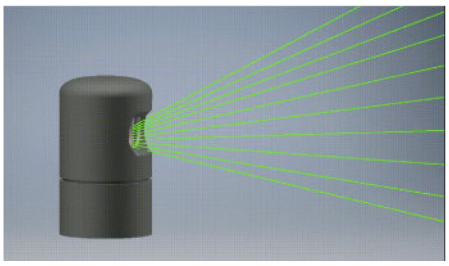
Tier IV Intelligent Vehicle

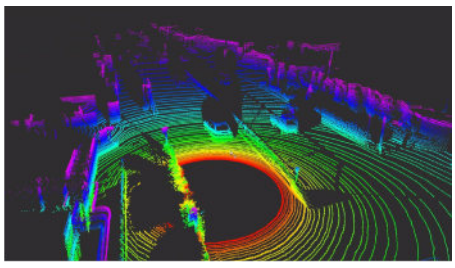
31

## 3D LiDAR Sensor Basics

**LiDAR**: Light Detection and Ranging
- Array of rotating lasers *or* one laser stirred by a mirror
- Infrared lasers, eye-safe (**1550** or **905 nm** wavelength)

**Data**: 3D information and reflectivity
- Time-of-flight used to determine **range** from sensor (1D)
- **Angle** of laser from origin [yaw, pitch] is known (2D)
  - We can calculate **3D coordinate** [x, y, z]
- 300k points (16 beams) to 1.3m points (64 beams) per second



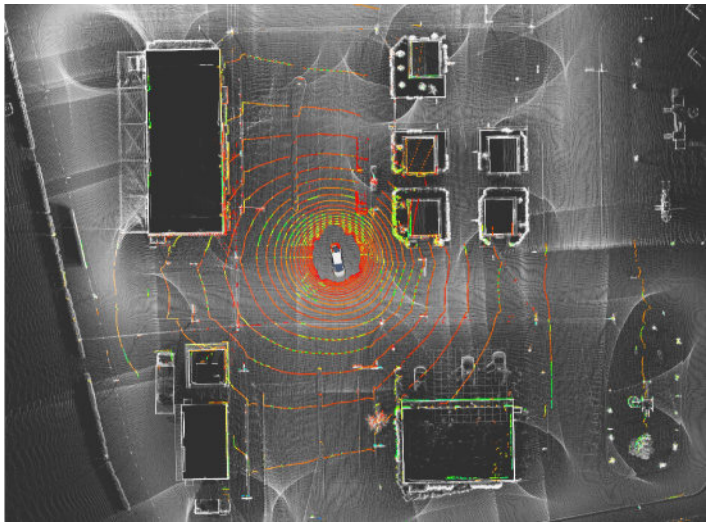


Tier IV Intelligent Vehicle

32

## 3D LiDAR Data – Point Clouds

**Reflectivity** can be estimated based on intensity of returned beam:

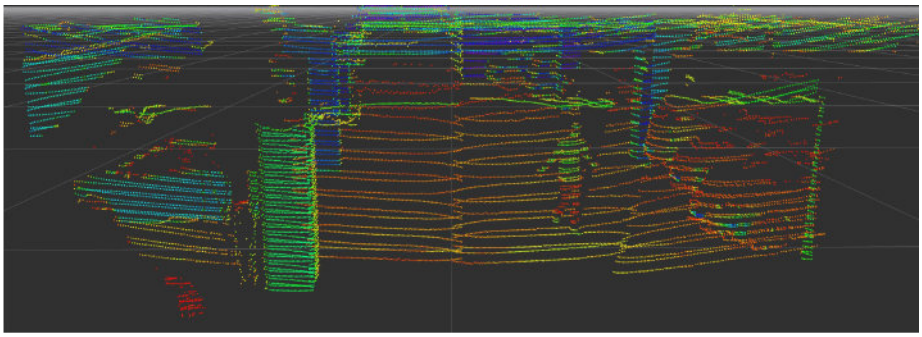


Tier IV Intelligent Vehicle

33

## 3D LiDAR Data – Point Clouds

Single-beam LiDAR with a rotating mirror produces sine-like scan lines.




Tier IV Intelligent Vehicle

34

## 3D LiDAR Sensor Specifications

**Range:**
- **Minimum range**: up to 1m; unreliable very close to sensor
- **Maximum range**, 200+ meters
- Precision: 1-2 cm normal error, progressively worse with range

Field of View (FOV):
- 360° degrees horizontal FOV, 0.1° to 1° resolution, depends on sensor rate
- Around 20° to 30° vertical FOV, resolution from 0.4° - 2.0° and adaptive.


Tier IV Intelligent Vehicle



## Why LiDAR for autonomous vehicles?

**LiDAR is an alternative – or complementary – sensor to cameras**

**Advantages:**
- **3D information** of raw environment; not a projection of the world.
- Much **larger FOV**, even vertical FOV sometimes.
- Infrared lasers **work just as well at night**
- Less susceptible to bad weather (rain, fog, light snow)
  - Recent multi-echo technology makes them reliable in rain

**Disadvantage:**
- **Sparse** at long range
- **Weaker texture** information
  - Not as dense as camera even at close range
  - Single-channel reflectivity, not as stable

- Susceptible to **interference** from other LiDAR or infrared sources
  - Less of a problem with latest technology
- Can be **bulky and power hungry**, more involved to install on a car
- Very expensive, thousands to tens of thousands.

Tier IV Intelligent Vehicle

## LiDAR on a vehicle

Installation on the top of the vehicle ensures best possible FOV, below is Velodyne 64-HDL, one of the largest models.



## LiDAR-based object detection

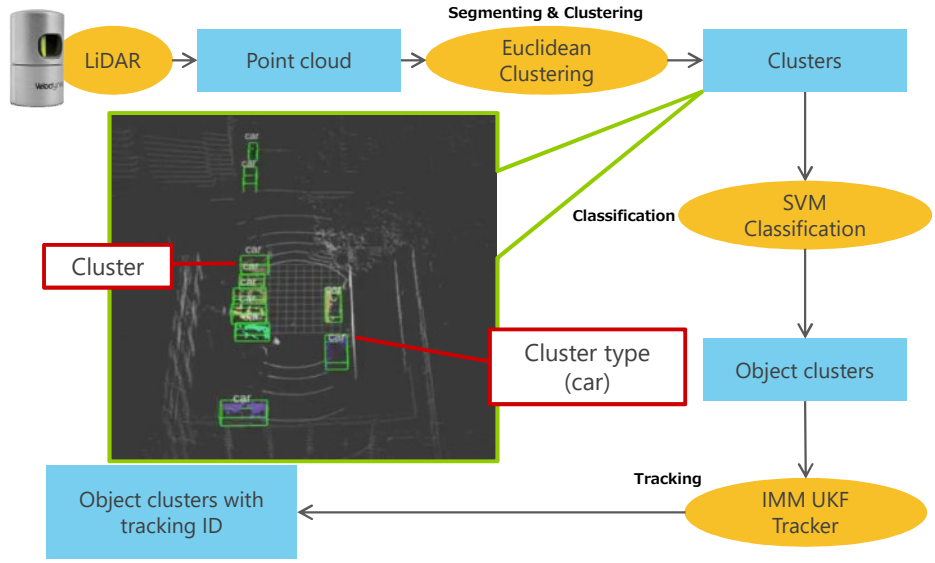
The goal of the lidar perception pipeline is:

1. **Segmenting & Clustering**
   Separating the data into **distinct segments**, and merging segments into **clusters** that correspond to the same object point cloud data into **clusters** with a bounding box; objects or obstacles.

2. Classification
   Determine the class of objects; car, cyclist, pedestrian or other.

3. Tracking
   Estimate the heading and velocity of relevant objects and track them over time.

## LiDAR Perception Pipeline






## Euclidean Clustering


Task to be accomplished during Euclidean Clustering:

1. Removes ground from data through ring filtering
2. Cluster remaining points
3. Visualize clusters

## Object Detection Message (Part 2)

Autoware uses one msg type at all steps of perception pipeline,
Euclidean clustering publishes: **/detection/lidar_objects**

DetectedObjectsArray.msg

```
std_msgs/Header          header
DetectedObjects[]        objects
```

DetectedObjects.msg

```
std_msgs/Header                 header
uint32                  id
string                          label
float32                 score   // Score as defined by the detection, Optional
std_msgs/ColorRGBA              color     // Define this object specific color

# 3D information
string                  space_frame   // 3D coordinate frame of the object, required if pose and dimensions are defined
geometry_msgs/Pose              pose
geometry_msgs/Vector3           dimensions
geometry_msgs/Vector3           variance
geometry_msgs/Twist             velocity
geometry_msgs/Twist             acceleration
sensor_msgs/PointCloud2         pointcloud
geometry_msgs/PolygonStamped    convex_hull
autoware_msgs/LaneArray         candidate_trajectories
bool                    pose_reliable
bool                    velocity_reliable
bool                    acceleration_reliable

# 2D
string                  image_frame // Image coordinate Frame,      Required if x,y,w,h defined
int32                   x           // X coord in image space(pixel) of the initial point of the Rect
int32                   y           // Y coord in image space(pixel) of the initial point of the Rect
int32                   width       // Width of the Rect in pixels
int32                   height      // Height of the Rect in pixels
float32                 angle       // Angle [0 to 2*PI), allow rotated rects

sensor_msgs/Image       roi_image
```

Tier IV Intelligent Vehicle



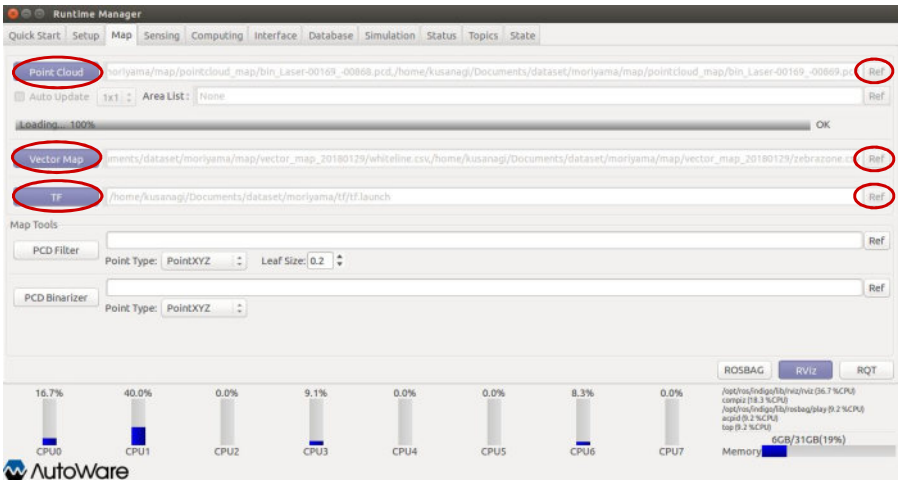
## Euclidean Clustering – Setup

Start, then pause the rosbag as before.


Tier IV Intelligent Vehicle

## Euclidean Clustering – Setup

Load the pointcloud map, vector map and TF






## Euclidean Clustering – Overview

Take a look at the Euclidean clustering – computing, lidar_detector
Click on the app.

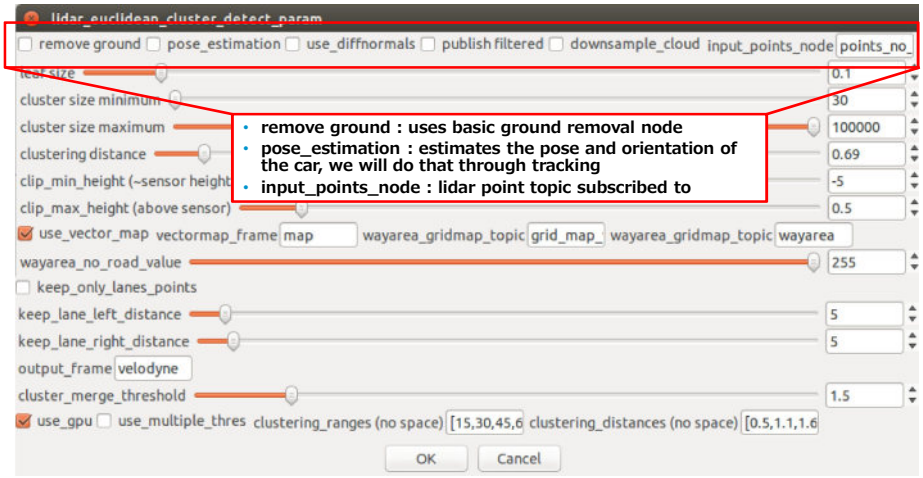
## Euclidean Clustering – Parameters

Several parameters can be adjusted based on setup



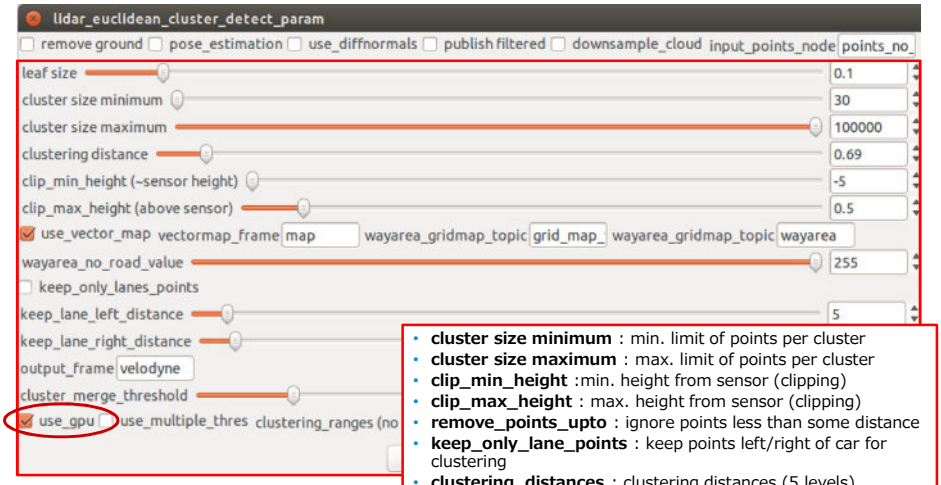

## Euclidean Clustering – Parameters

Several parameters can be adjusted based on setup



## Euclidean Clustering – Parameters

Several parameters can be adjusted based on setup



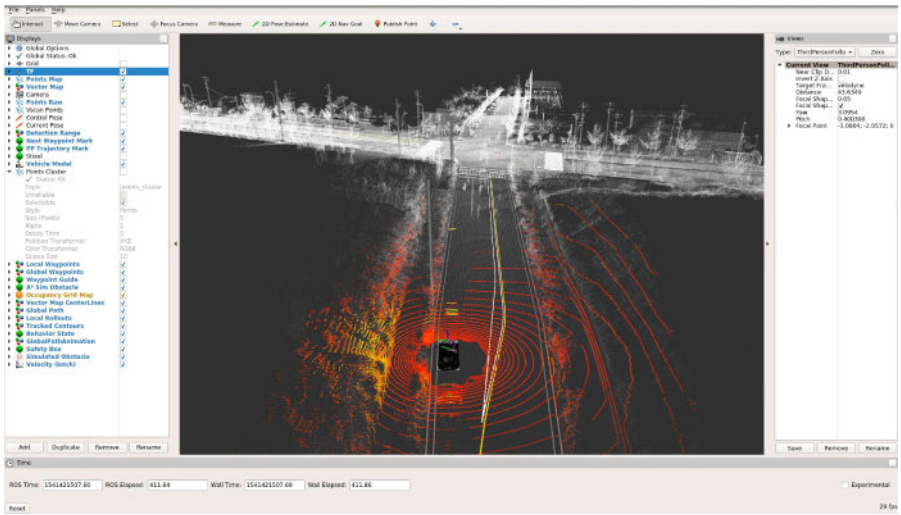

- **cluster size minimum** : min. limit of points per cluster
- **cluster size maximum** : max. limit of points per cluster
- **clip_min_height** :min. height from sensor (clipping)
- **clip_max_height** : max. height from sensor (clipping)
- **remove_points_upto** : ignore points less than some distance
- **keep_only_lane_points** : keep points left/right of car for clustering
- **clustering_distances** : clustering distances (5 levels)
- **clustering_thresholds** : threshold for each distance (5 levels)

Tier IV Intelligent Vehicle

47

## Euclidean Clustering – Raw

Euclidean Clustering on the entire pointcloud is expensive and inaccurate

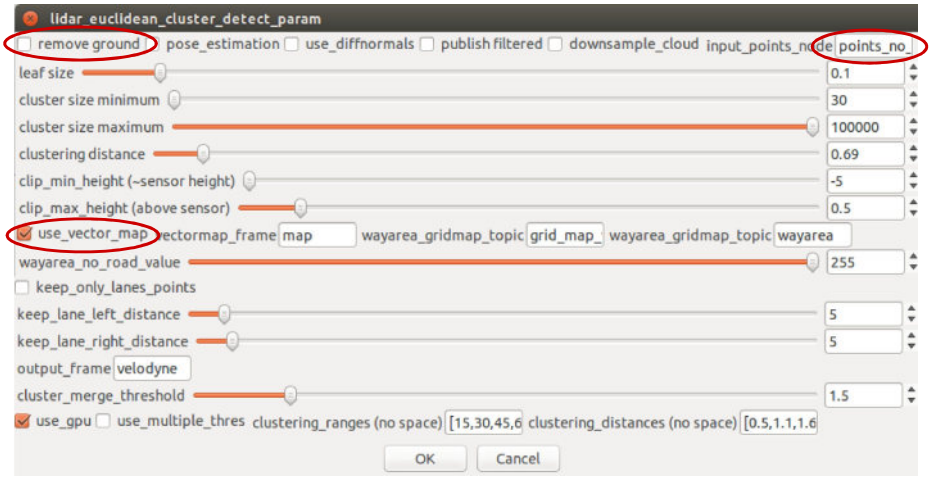


Tier IV Intelligent Vehicle

48

## Euclidean Clustering – Filtering

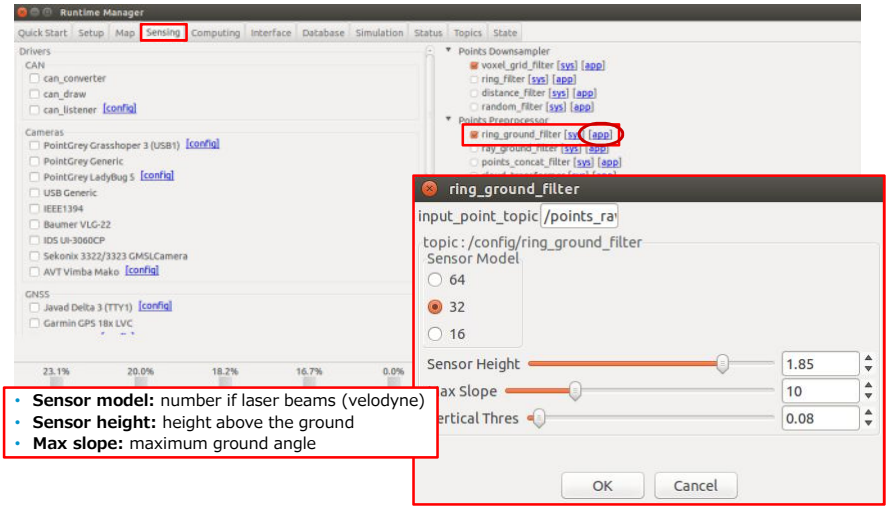
We can easily filter a lot of the data, making it more accurate.






## Ring Ground Filtering

A better ground filtering procedure has been implemented in Autoware

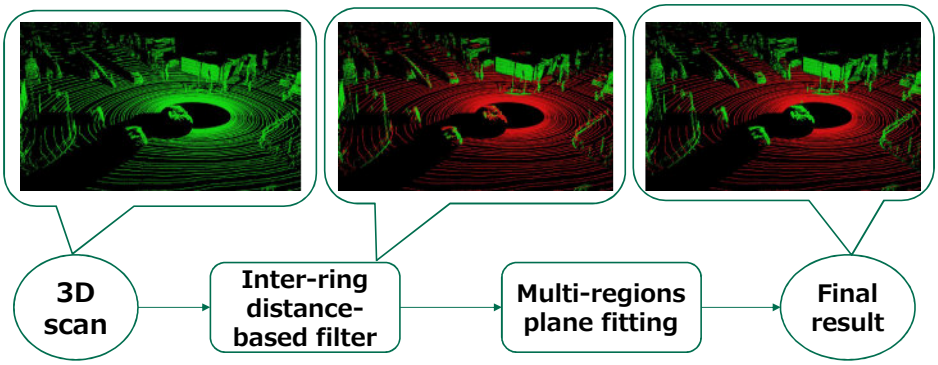



- **Sensor model:** number if laser beams (velodyne)
- **Sensor height:** height above the ground
- **Max slope:** maximum ground angle

## Ring Ground Filtering

Ground filtering consists of two step:
- Ring distance-based filtering
- Multi-region plane fitting

Processing pipeline:

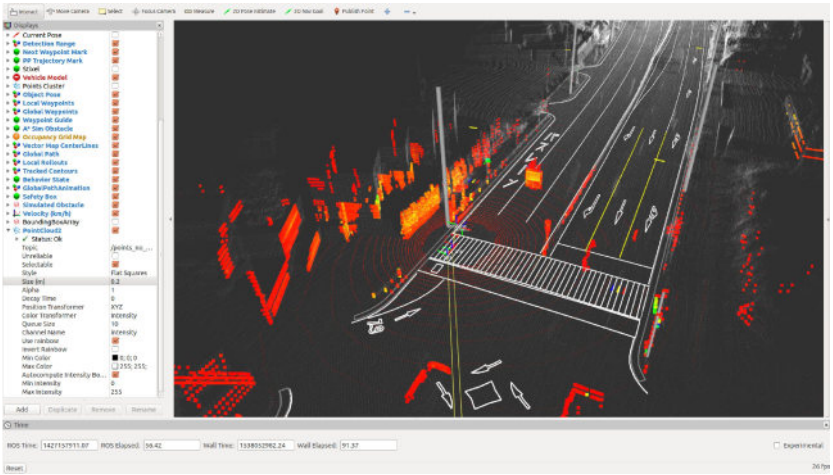



Tier IV Intelligent Vehicle

51

## Ring Ground Filtering

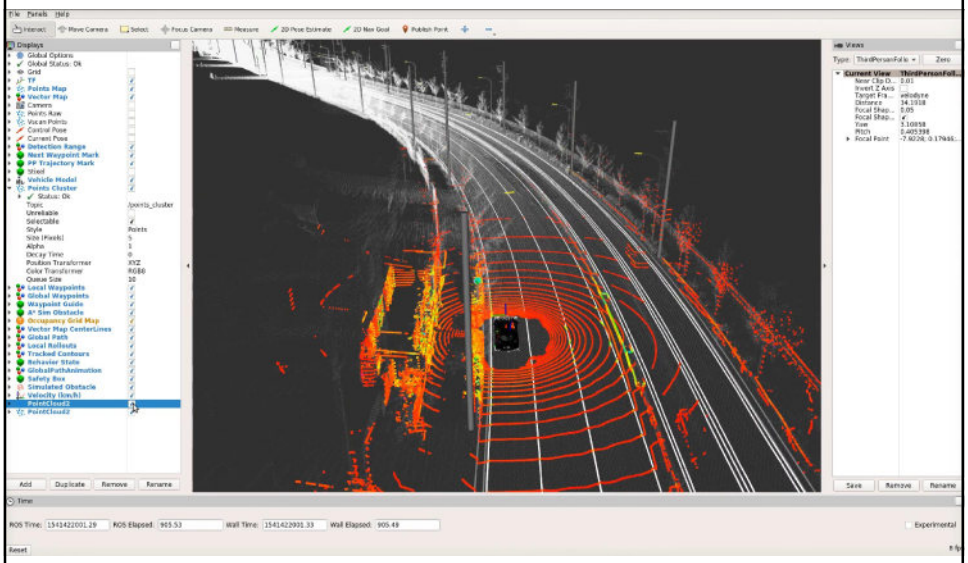
Ring ground filter publishes two topics you can visualize in RViz:
- /points_ground : points determined to be ground points
- **/points_no_ground** : points determined not to be ground points



Tier IV Intelligent Vehicle

52

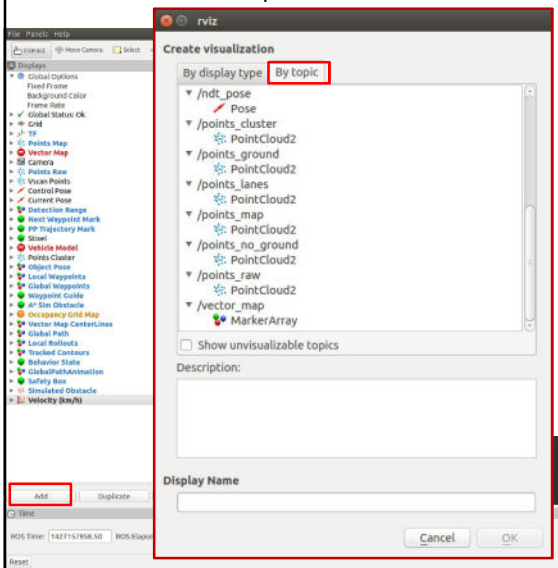
## Ring Ground Filtering





## Point Cloud Visualization

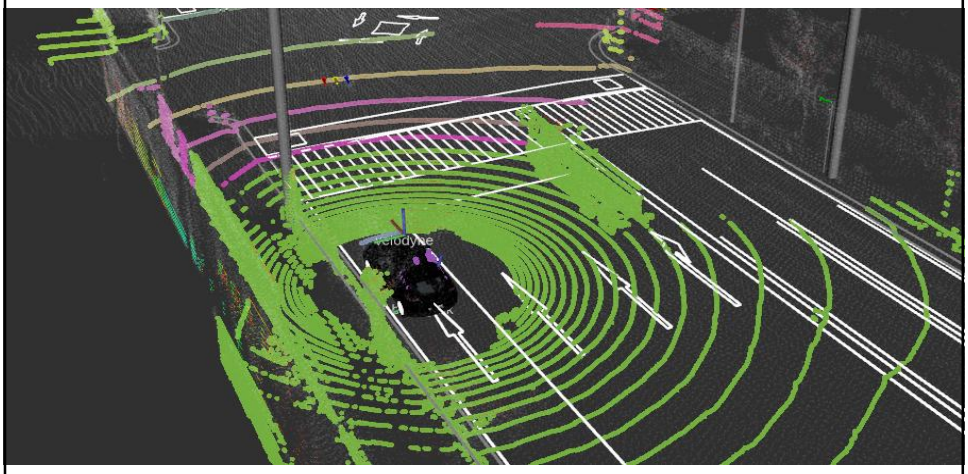
Let's see colored points of each cluster from **/points_cluster** .




- **/points_cluster: output from the Euclidean clustering**
- /points_ground: output of ground removal
- /points_no_ground: output of ground removal, used by Euclidean clustering
- /points_lanes: Euclidean clusters in lanes only
- /points_raw: all data from lidar sensor
- /points_map: NDT pointcloud map

## Impact of Ground Filtering

Comparing with and without ground filtering

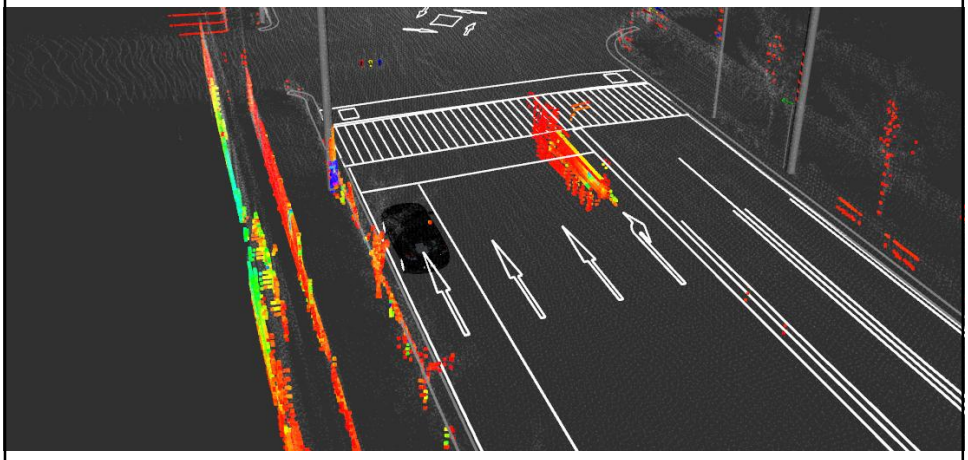



Tier IV Intelligent Vehicle

55

## Impact of Ground Filtering

Comparing with and without ground filtering

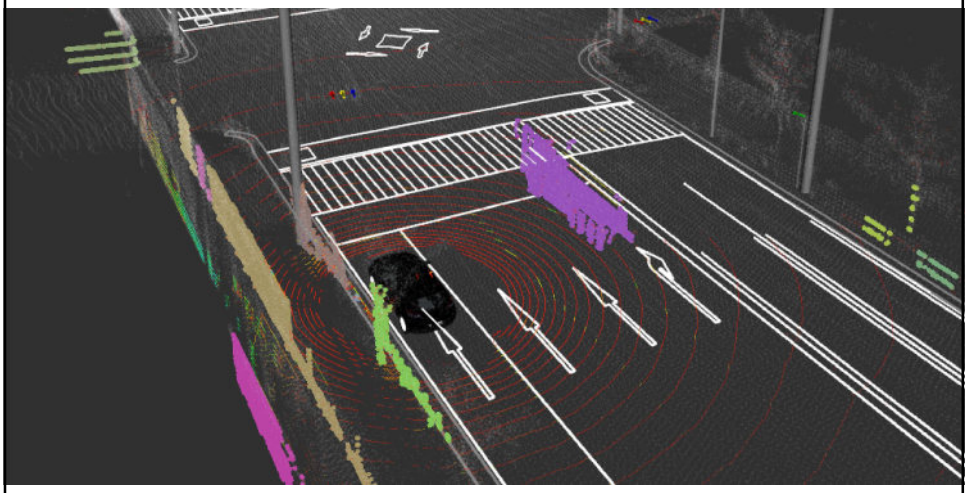


Tier IV Intelligent Vehicle

56

## Impact of Ground Filtering

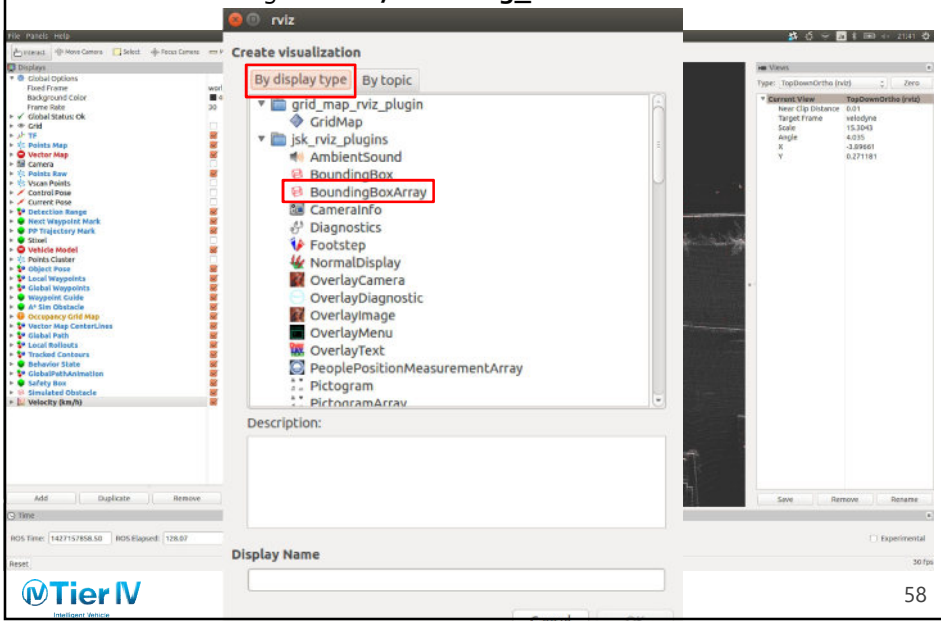
Comparing with and without ground filtering





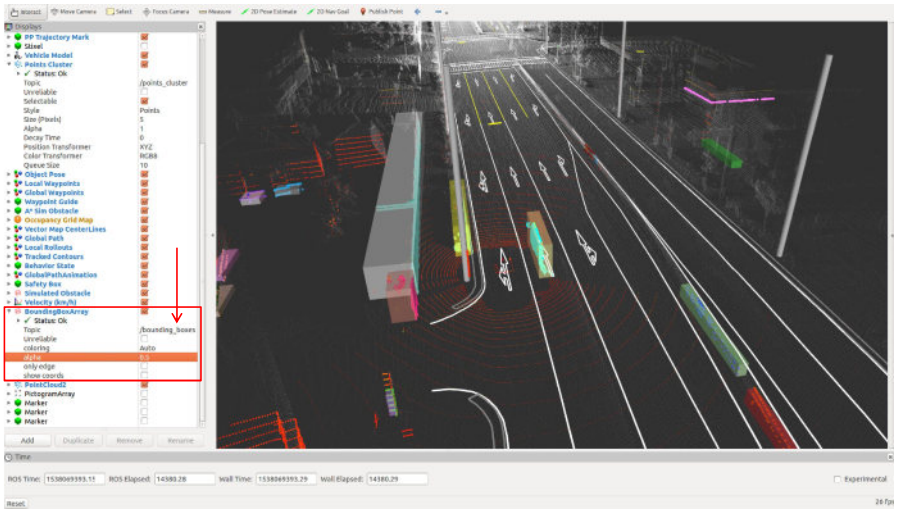
## Bounding Box Visualization

Let's see bounding boxes : **/bounding_boxes**



## Bounding Box Visualization

Let's see bounding boxes : **/bounding_boxes**

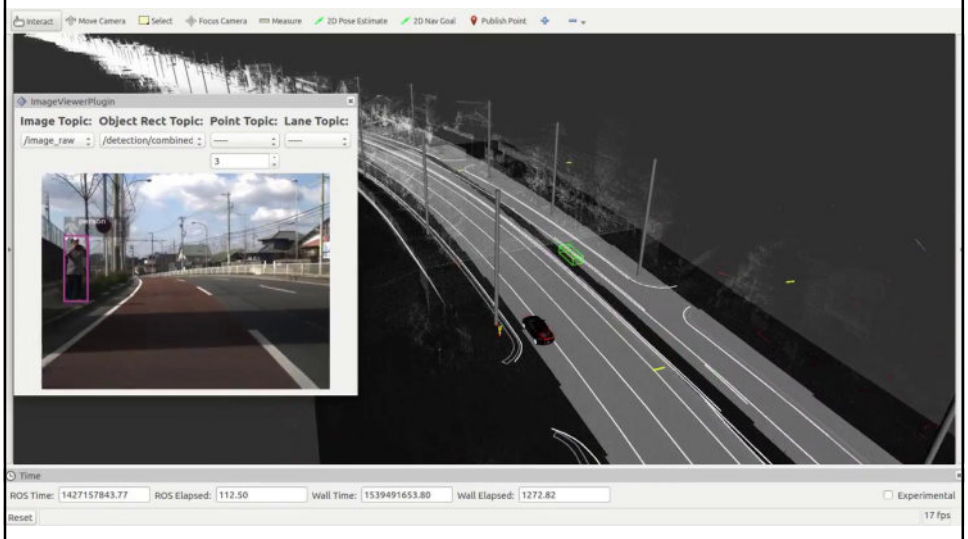


Tier IV Intelligent Vehicle

59

## Vector Map Filtering

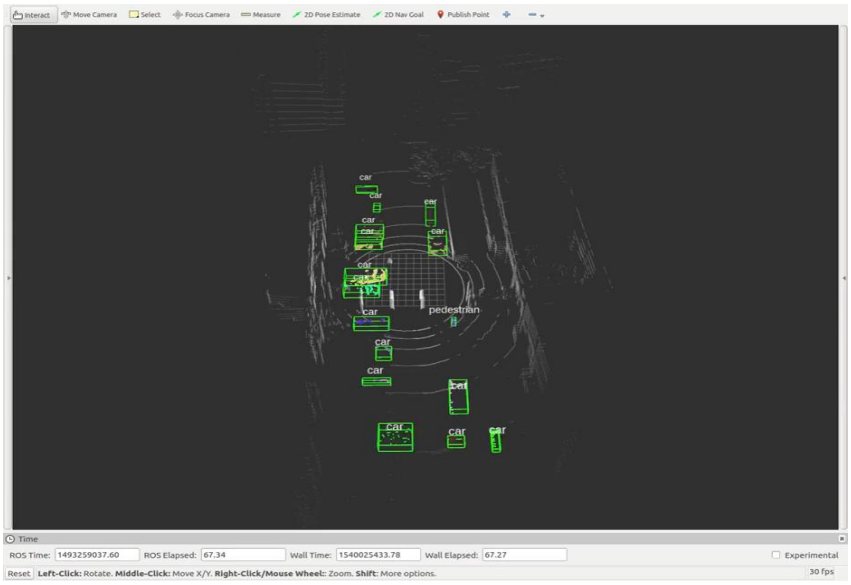
We can also use the Vector Map to filter buildings




Tier IV Intelligent Vehicle

60

## Step 2 - Euclidean Clustering




Tier IV

61

## Tracking

Now that we have point clusters, we'd like to classify and track them.

- **lidar_svm_detect** : svm classification, **assigns cluster ids**
- **lidar_kf_track** : Kalman filter based tracking, simple and quick
- **lidar_imm_ukf_pda_track** : unscented Kalman filter based tracking, more advanced and reliable


Tier IV

62

## Tracking – IMM-UKF Tracker

IMM-UKF is more sophisticated, keeps track of objects even when they disappear for a short time, making it more stable.

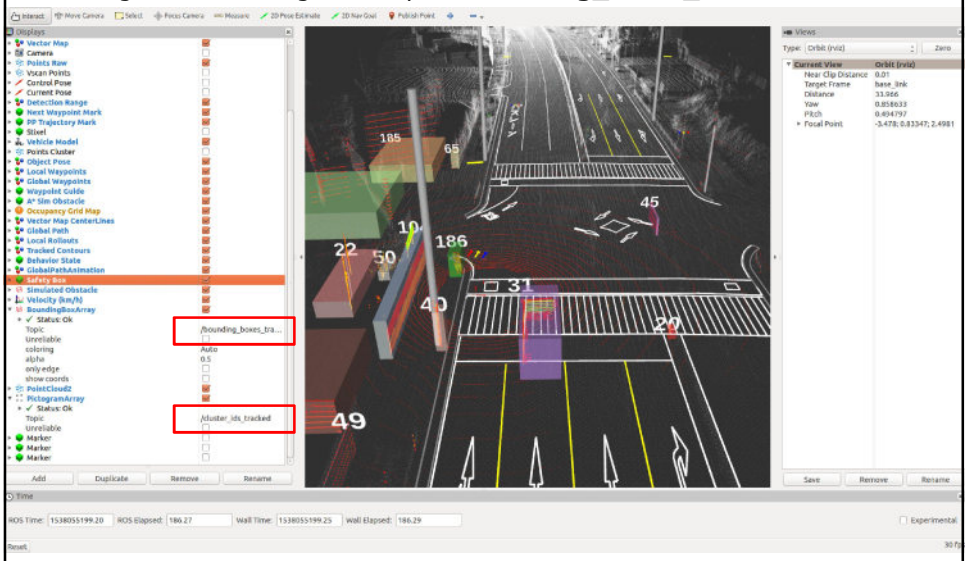



## Tracking – IMM-UKF Tracker

IMM-UKF is more sophisticated, keeps track of objects even when they disappear for a short time, making it more stable.

## Tracking – IMM-UKF Tracker

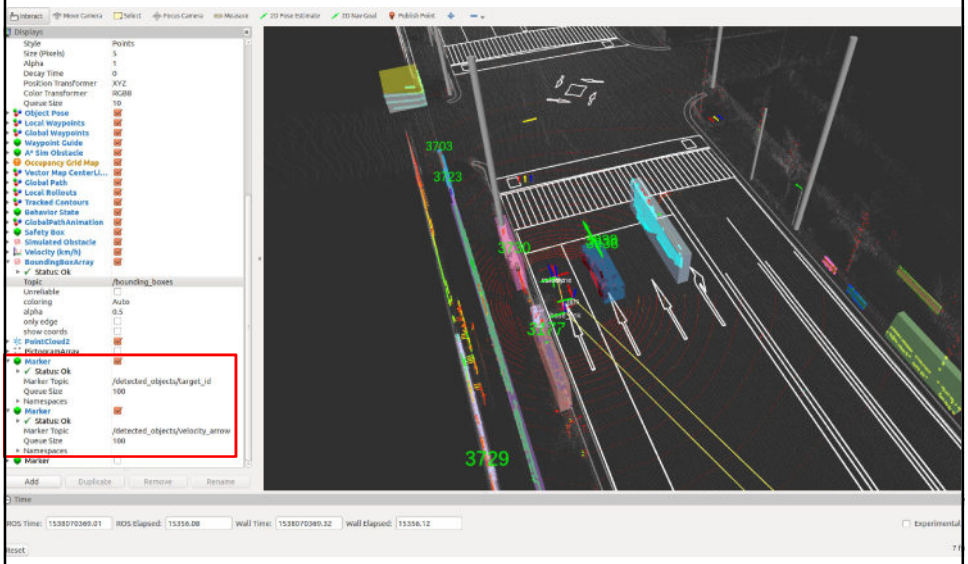
Change the BoundingBoxArray to **/bounding_boxes_tracked**




Tier IV Intelligent Vehicle

65

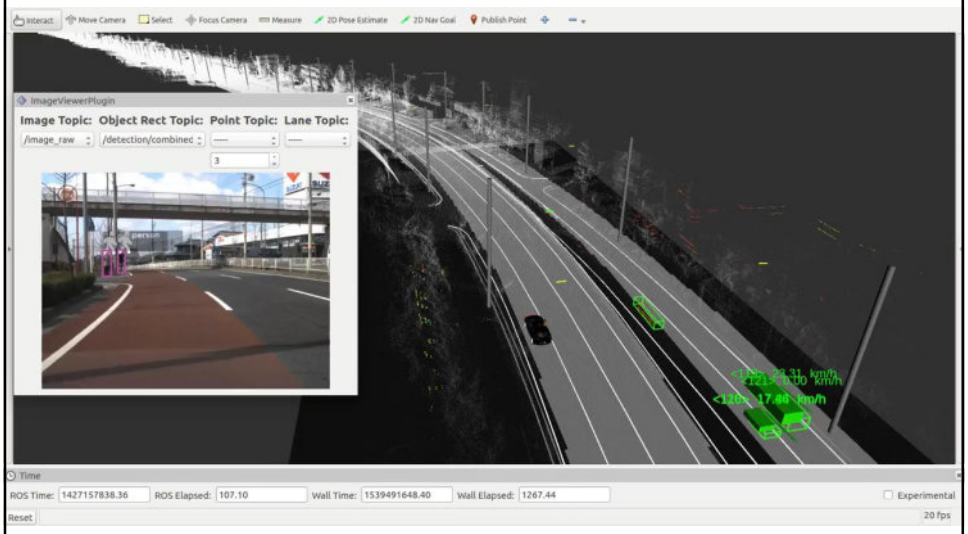
## Tracking – IMM-UKF Tracker




Tier IV Intelligent Vehicle

66

## Tracking – IMM-UKF Tracker

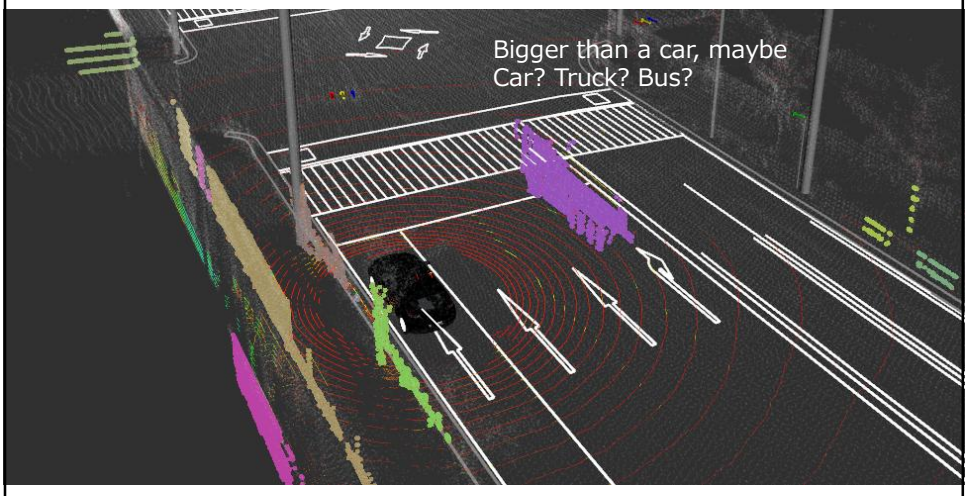



Tier IV

67

## LiDAR Classification is Difficult!

Differentiating types of car, and pedestrians from cyclists is challenging

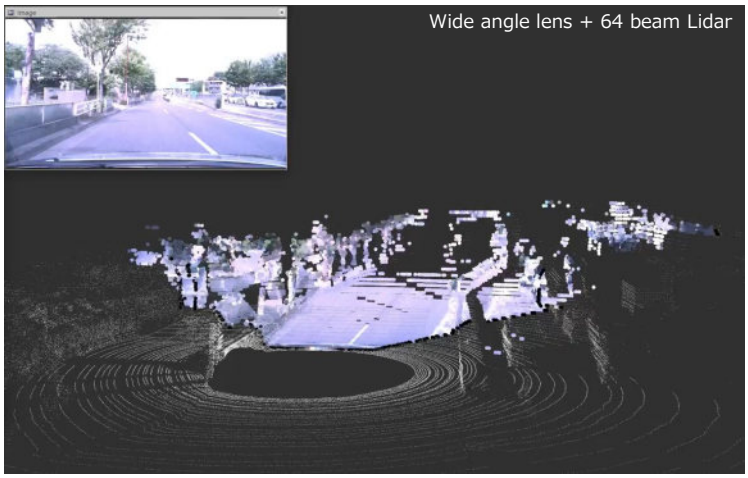



Tier IV

68

## Camera-LiDAR Fusion – The Best of Both Worlds

Each sensor modality has issues, but they complement eachother:
- Camera provide **dense, colorful** information,
- LiDAR provides very **accurate range** information



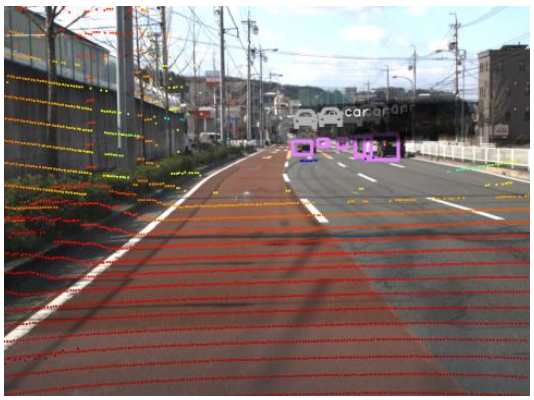

Tier IV Intelligent Vehicle

69

## Camera-LiDAR Fusion - Classification

Sensor fusion comes at a cost: we need **sensor calibration**:
1. Camera Intrinsic Calibration (image rectification)
2. Lidar-Camera Extrinsic Calibration

We need to be able to **correlate lidar data and image data**



Tier IV Intelligent Vehicle

70

## Camera Intrinsic Calibration

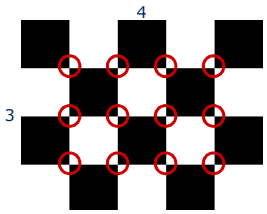
This tool is not yet available in GUI, must launch from terminal.

- cd ~/Autoware/ros/
- source devel/setup.bash
- rosrun autoware_camera_lidar_calibrator cameracalibrator.py --square 0.1 --size MxN image:=/image_raw

There are several parameters based on your setup:

- --square is the size of chessboard square in meters
- --size is chessboard size, as in number of inner square corners
- image:= image topic (needs to be published)
- --detection cv2 or matlab, matlab is more reliable but cv2 is open-source.

If launched correctly, the GUI should pop-up with image being displayed.



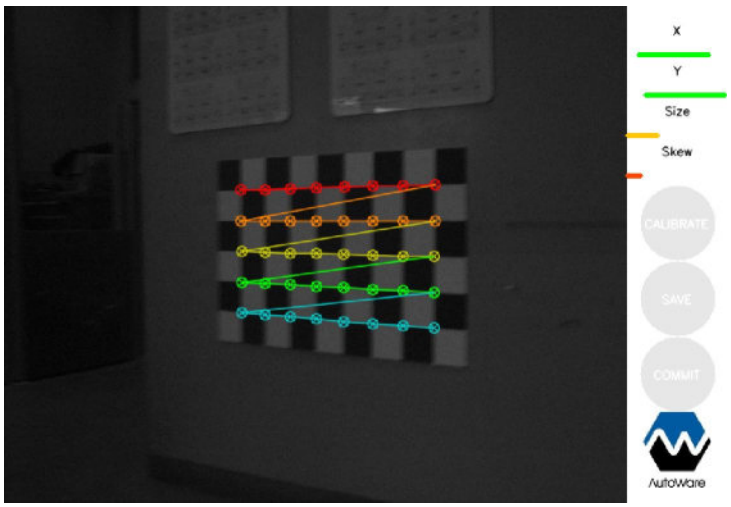

--size 4x3

Tier IV

71

## Camera Intrinsics Calibration

Intrinsics camera calibration GUI




Tier IV

72

## Camera Extrinsics – Launch Calibration

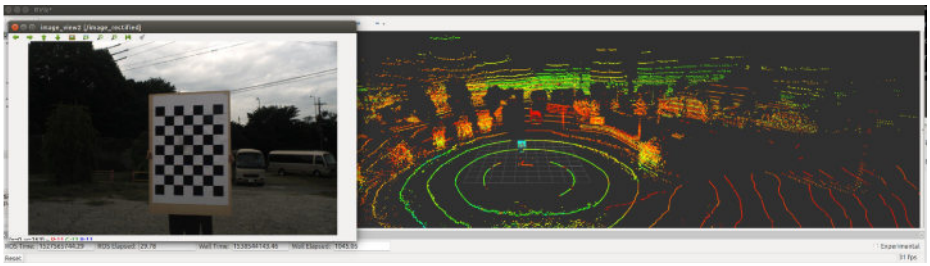
Launching the extrinsic calibration node is also through command line:

- roslaunch autoware_camera_lidar_calibrator camera_lidar_calibration.launch intrinsics_file:=camera_intrinsic_calibration.yaml image_src:=/image

Some parameters to specify:

- instrinsic_file: the output of the intrinsic calibration
- image_src: the image topic name

Then you can set up Rviz to see the image and pointcloud side by side.

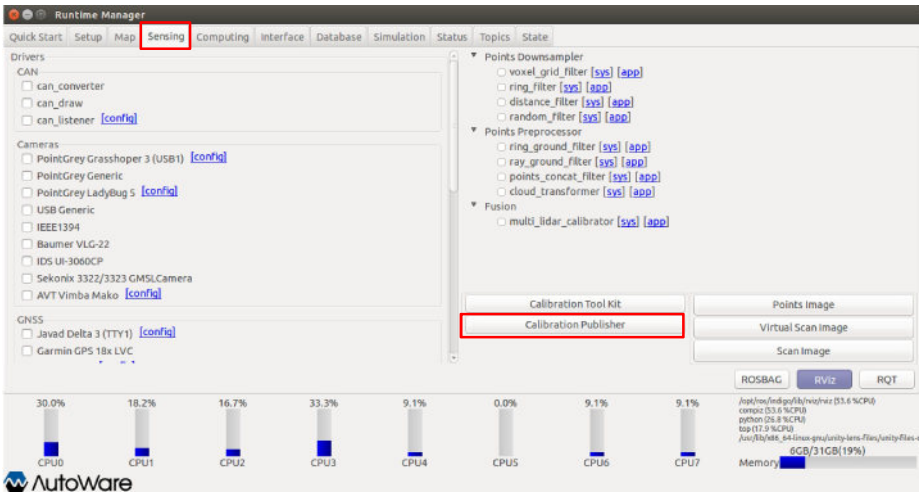


Tier IV



## Step 1 – Load in Calibration File

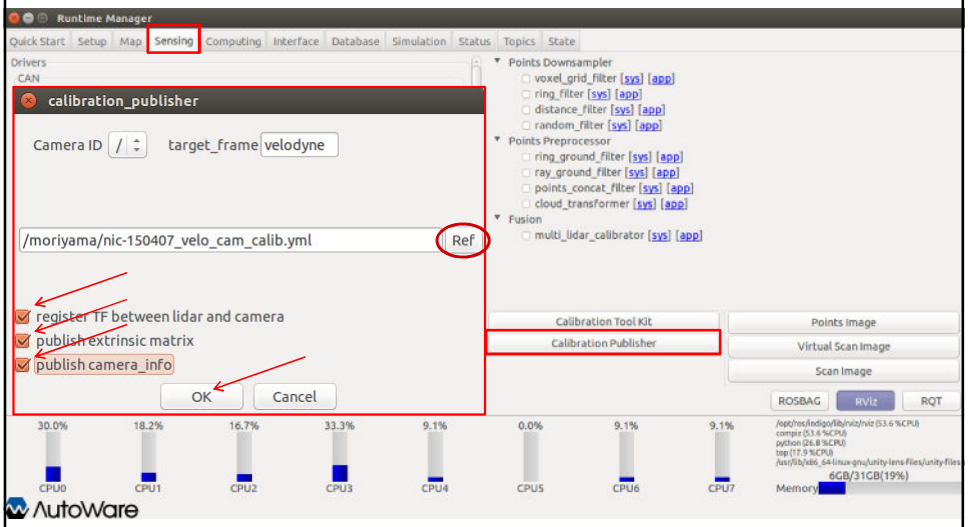
Go to the Sensing tab and click on the Calibration Publisher




Tier IV

## Step 1 – Load in Calibration File

Load in the calibration file, click all checkboxes and click OK

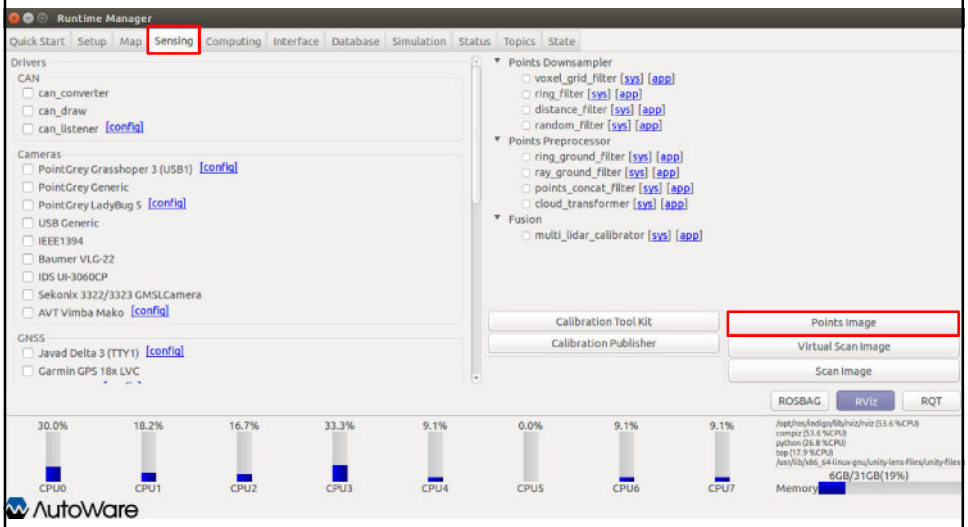



Tier IV

75

## Step 2 – Create the points_image

Click the Points Image button and OK

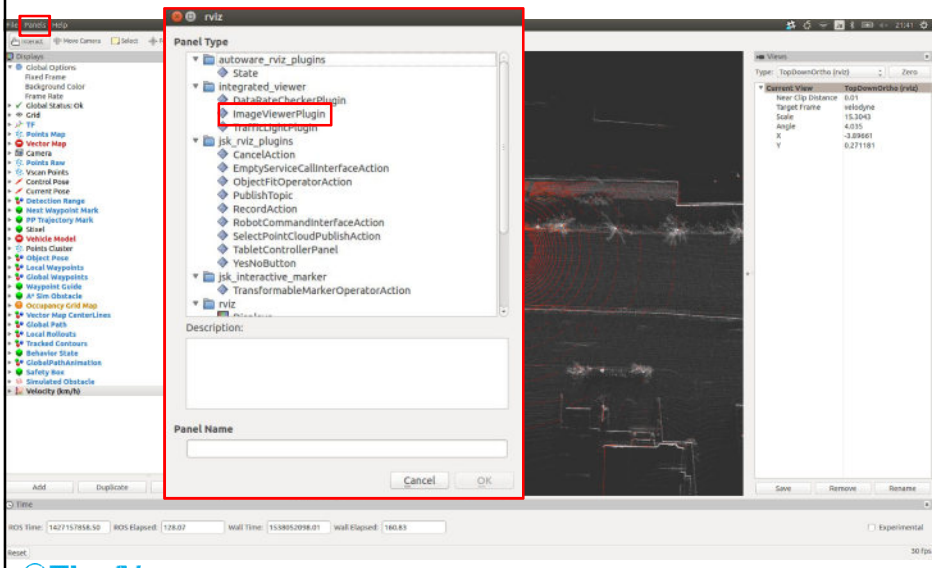



Tier IV

76

## Step 3 – Visualize LiDAR points in Camera image

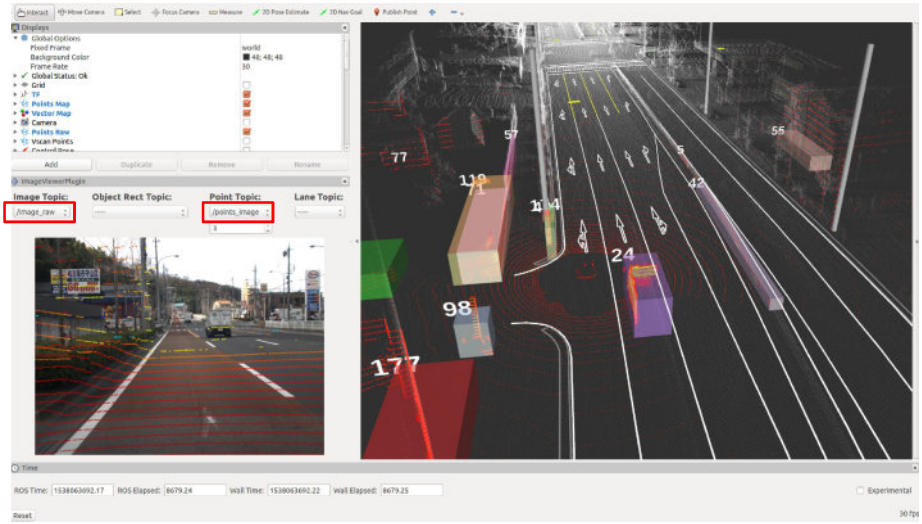
On RVIZ taskbar click Panels -> Add New Panel -> ImageViewerPlugin






## Step 3 – Visualize LiDAR points in Camera image

Set the image topic to **/image_raw** and point topic to **/points_image**



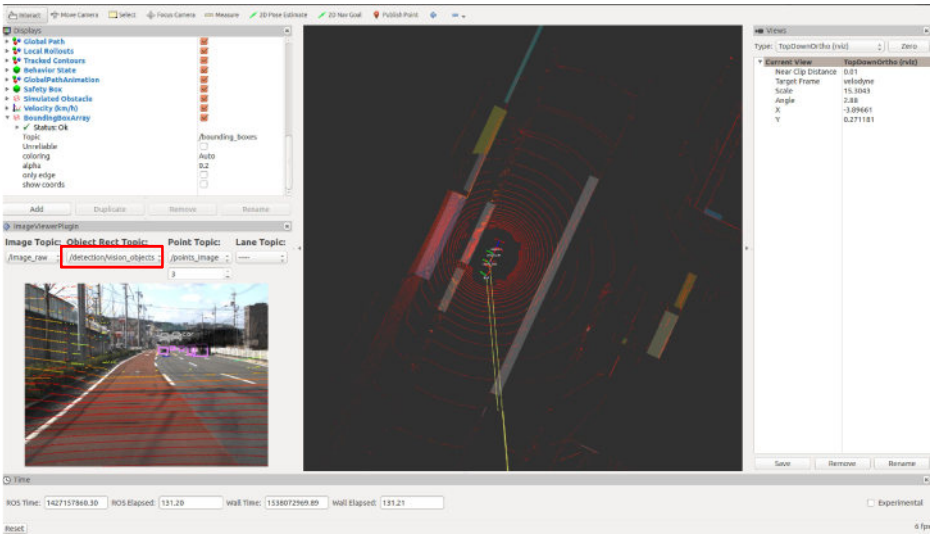
## Step 4 – Launch Image-based Classification

In Computing -> vision_detection, launch **vision_darknet_yolo3**




## Step 4 – Launch Image-based Classification

You can see the Yolo3 detection in the ImageViewerPlugin



## Step 5 – Launch LiDAR tracking

IMM UKF tracking should still be running,

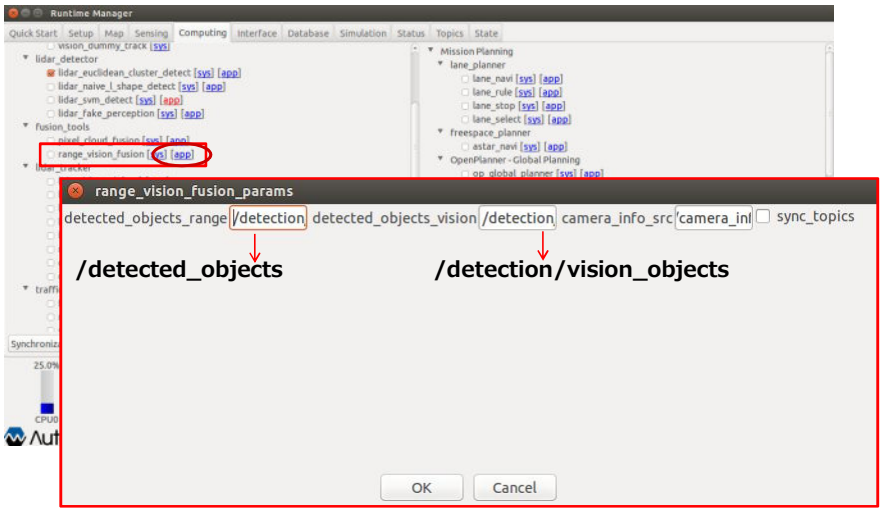



## Step 6 – Launch Image-Lidar Fusion Classification

In Computing -> vision_detection, launch **vision_darknet_yolo3**

## Step 6 – Launch Image-Lidar Fusion Classification

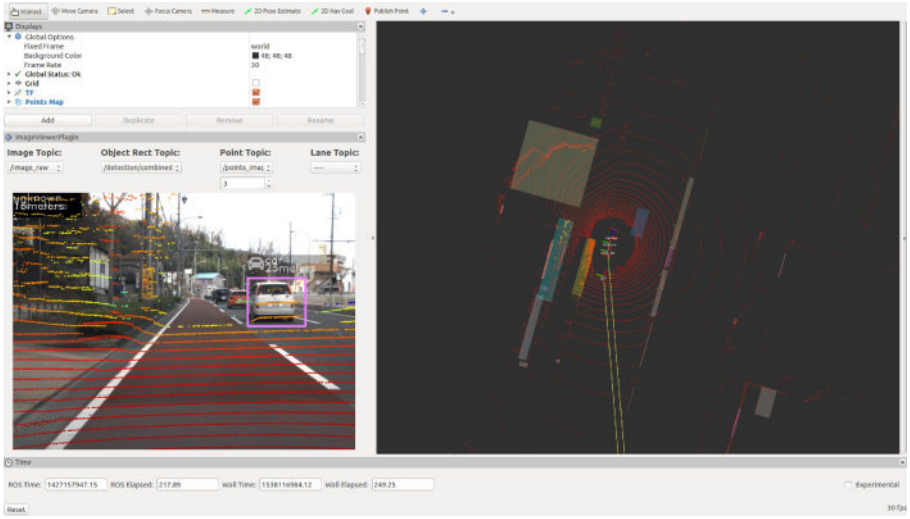
In Computing -> vision_detection, launch **vision_darknet_yolo3**



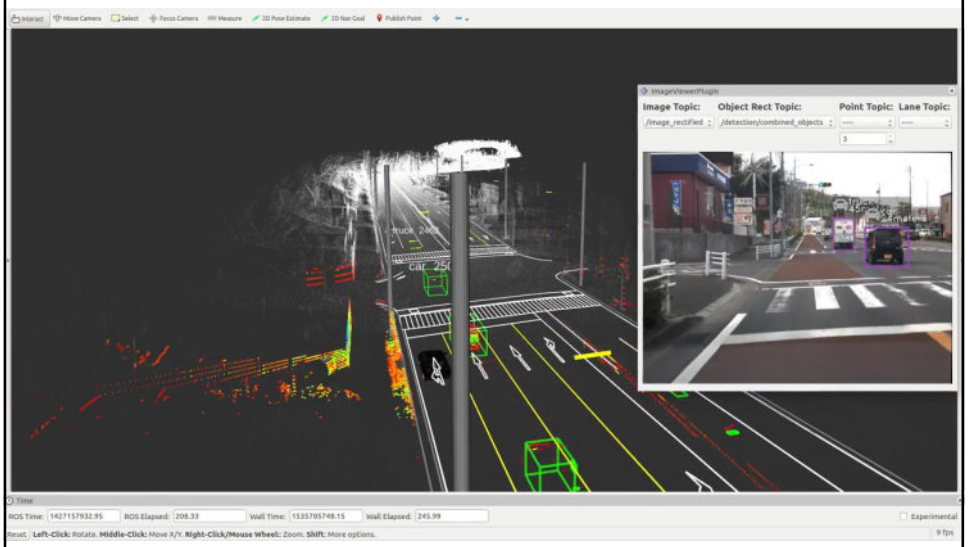



## Step 6 – Launch Image-Lidar Fusion Classification

Change the topic in ImageViewerPlugin to **/detection/combined**

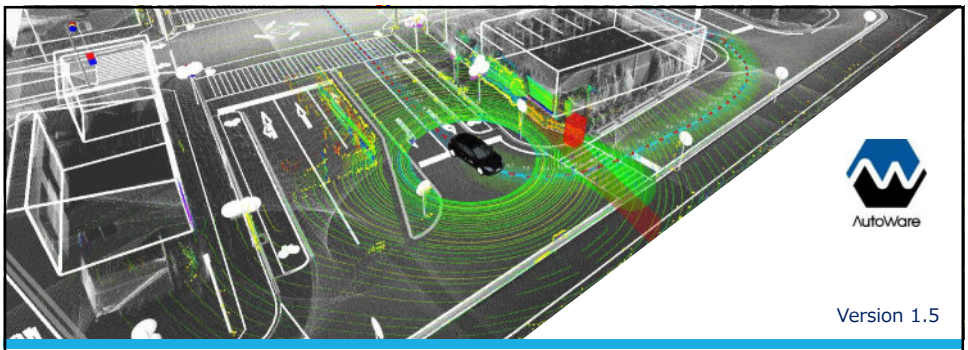


## Step 6 – Launch Image-Lidar Fusion Classification




Tier IV Intelligent Vehicle

85



Autoware

Version 1.5

Autoware Hands-on Experience

## Chapter 5：Path Generation and Path Planning

1. Path Generation

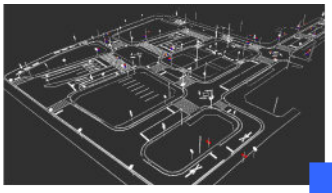
Tier IV Intelligent Vehicle

86

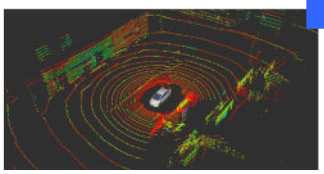
## Path Generation – Overview

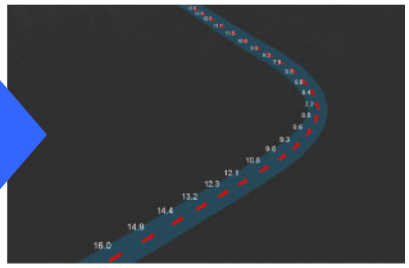
- From Vector Map and localization information, a path is generated
- If Vector Map is used, a path is generated from the centerline information the road
- If no Vector Map is available, a path is generated from log data
- The path is output as data string with coordinate, direction and velocity information

Vector Map



Position and velocity information from localization

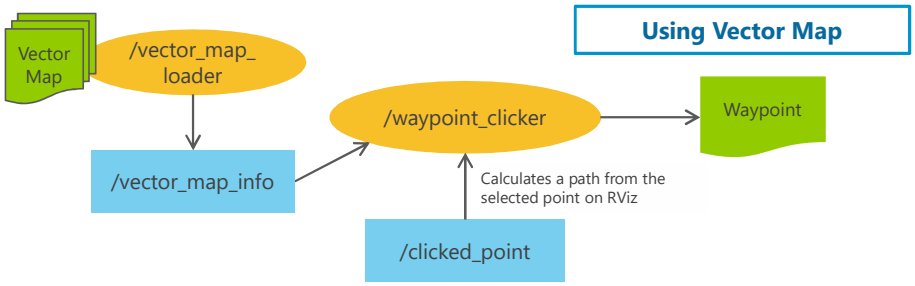


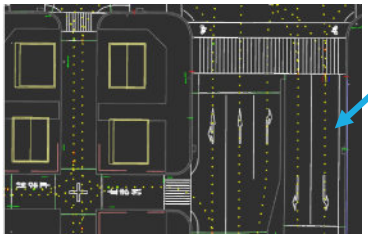


Path (Waypoint)



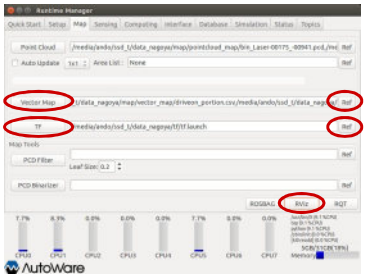
## Path Generation – Structure



Create path from lane information in vector map

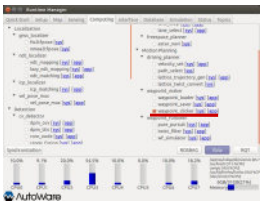
## Path Generation Using Vector Map – Step (1/2)



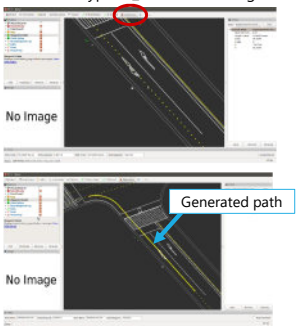
1. Vector Map loading
   - A) Clicking [Ref] on [Vector Map] in [Map] tab, specify a Vector Map file
   - B) Click [Vector Map] button
2. Specify TF to indicate Vector Map position
   - A) Clicking [Ref] on [TF] in [Map] tab, select a TF file according to the Vector Map position.
   - B) Click [TF] button
     (this publishes TF between world frames and map frames)
3. Launch RViz
   - A) Click [Rviz]
   - B) Pressing either [File]->[Open Config], or [Ctrl + O], specify [Autoware/ros/src/.config/rviz/default.rviz]
   - C) Confirm that Vector Map is displayed



## Path Generation Using Vector Map – Step (2/2)



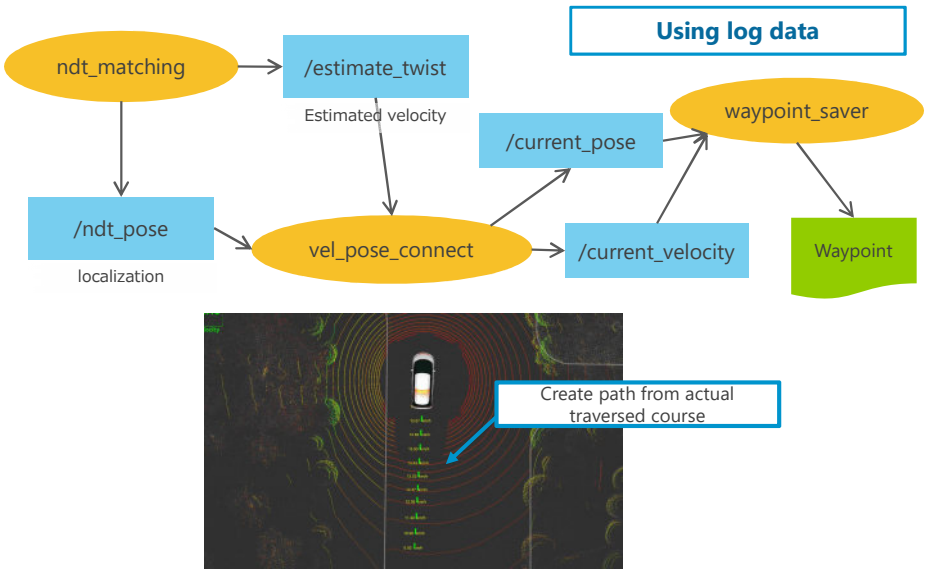
4. waypoint_clicker launching




5. Path generation by [Publish Point] in RViz

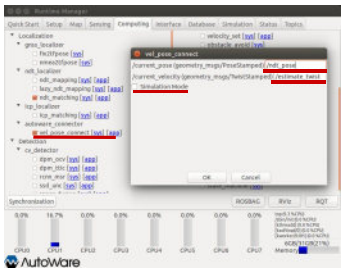
4. Launch "waypoint_clicker"
   - A) Launch [waypoint_clicker] checking (☑) the box in [Computing] tab
   - ➢ Path, velocity and path to save the file can be specified
5. Path generation by [Publish Point] on RViz
   - A) Confirm that multiple yellow points are being displayed on RViz
   - B) Click [Publish Point] in the top of RViz, click the yellow points displayed on RViz
   - ➢ Zooming and moving the mouse over the yellow points in RViz, these points can be clicked.
   - C) Clicking the yellow points on the same lane, yellow line is generated
   - ➢ If the selected yellow points are linkable, even long paths can be generated
   - ➢ Note that, if green or red points are located on intersections, path generation may not success without clicking them
   - D) Checking off the box of [waypoint_clicker] in [Computing] tab, a path file is generated in the directory specified in [app]

## Path Generation Using Vector Map – Step (2/2)






## Path Generation by ROSBAG – Step (1/2)




2. vel_pose_connect launching

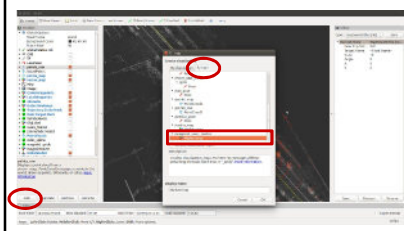

3. waypoint_saver launching

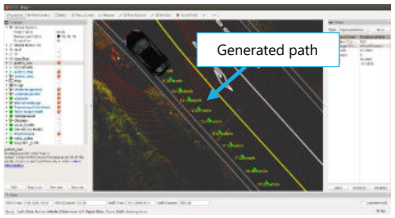
1. ROSBAG playback and localization
   - ➢ Refer to localization

2. "vel_pose_connect" launching
   - A) Launch [vel_pose_connect] by checking (☑) the box in [Computing] tab
   - ➢ Confirm that the box of [Simulation Mode] in [app] is not checked, and that [ndt_pose] and [/estimate_twist] are defined as inputs

3. "waypoint_saver" launching
   - A) Click on the app button of [waypoint_saver], and specify the filename
   - B) Launch [waypoint_saver] by checking (☑) the box in [Computing] tab

## Path Generation by ROSBAG – Step (2/2)



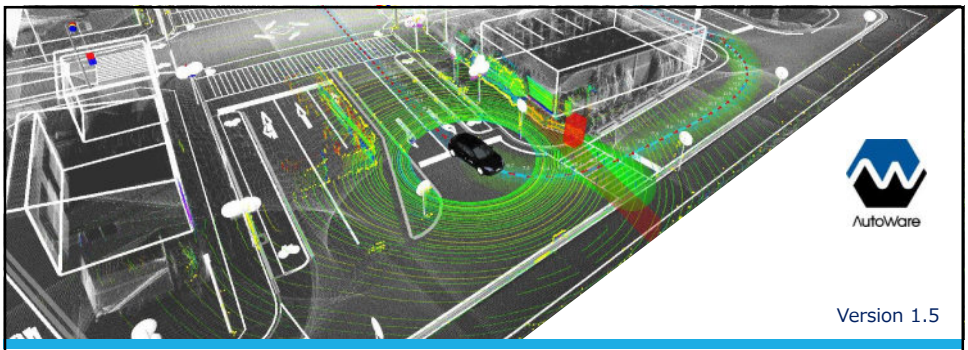



4. /waypoint_saver_marker on RViz displaying

4. Displaying "/waypoint_saver_marker" on RViz
   - A) Click [add] button in the bottom of the left topic list in RViz
   - B) Click [/waypoint_saver_marker] -> [MarkerArray] from [By Topic] tab of the displayed window.
   - C) Confirm the marker logging waypoint in RViz is being displayed

5. Ending "waypoint_saver"
   - A) End path generation by unchecking [waypoint_saver] on [Computing] tab in [Runtime Manager]
   - ➢ The file is saved into the directory specified in [app]





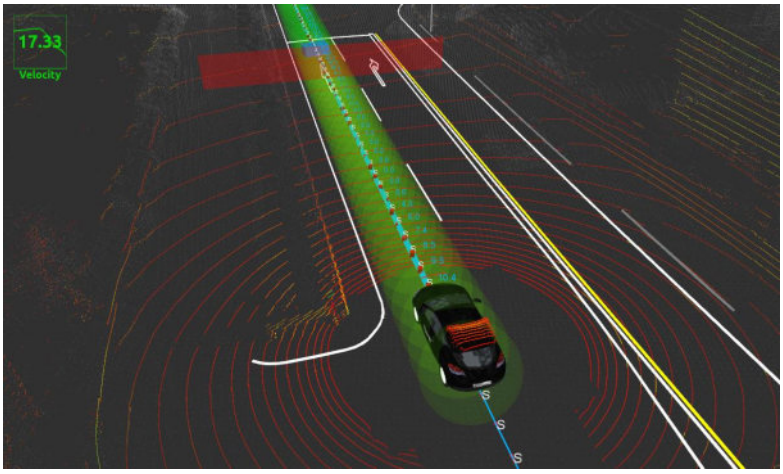

Autoware Hands-on Experience

# Chapter 4 : Path Generation and Path Planning

## 2. Path Planning

## Path Planning – Overview

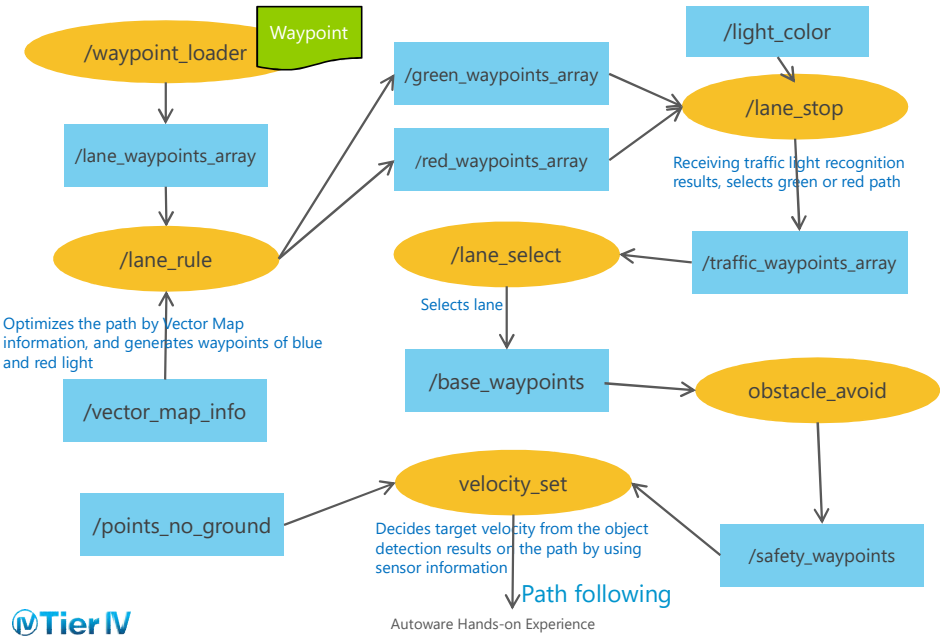
- Loads a previously saved path file
- Optimizes velocities on the path with high definition map information, etc.
- Selects a lane, if there are multiple lanes



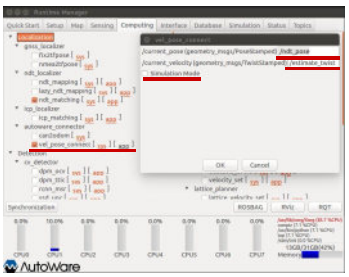



## Path Planning – Structure

※ Some topics are omitted



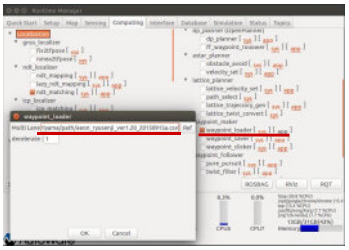
## Path Planning – Step (1/4)



2. vel_pose_connect launching

1. ROSBAG playback and localization
   - ➢ Refer to localization chapter

2. "vel_pose_connect" launching
   - A) Launch [vel_pose_connect] by checking (☑) the box in [Computing] tab
   - ➢ Confirm that the box of [Simulation Mode] in [app] is not checked, and that [ndt_pose] and [/estimate_twist] are defined as inputs

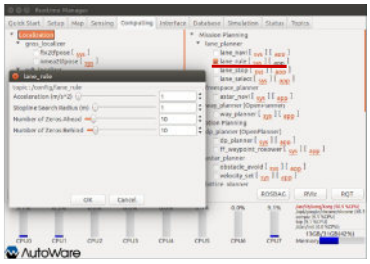


3. waypoint_loader launching

3. Path loading using waypoint_loader
   - A) Specify the path file in [app]
   - ➢ Several files can be specified, they'll be combined into one trajectory



## Path Planning – Step (2/4)



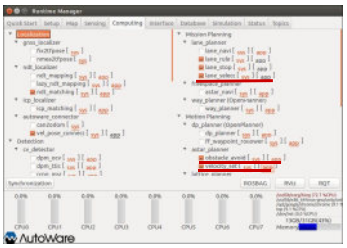
4. lane_rule launching

4. Launching "lane_rule"
   - A) Launch [lane_rule] by checking (☑) the box
   - ➢ [Number of Zeros Ahead] and [Number of Zeros Behind] define the number of waypoint (velocity 0) at stop line when traffic light is red.

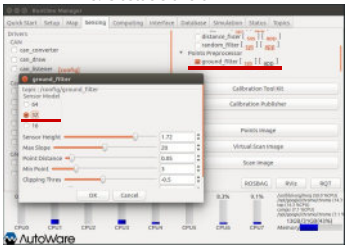
5. lane_stop launching

5. Launching "lane_stop"
   - A) Launch [lane_stop] by checking (☑) the box
   - ➢ Check [Use traffic light recognition], this node automatically switches path between blue and red path by receiving traffic light recognition node results.

## Path Planning – Step (3/4)



6. Lane select
7. Velocity set
8. Obstacle avoid



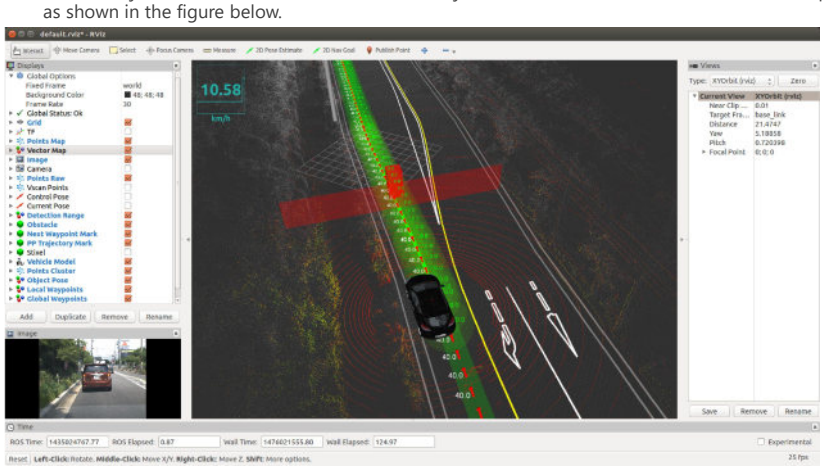
9. Ground filter

6. Launching "lane select"
   - A) Launch [lane_select] by checking (☑) the box
     - ➢ Following paths can be selected in [app]
     - ➢ If load only one path by [waypoint_loader], select the corresponding path.
7. Launching "velocity set"
   - A) Launch [velocity_set] by checking (☑) the box
8. Launching "obstacle avoid"
   - A) Launch [obstacle_avoid] by checking (☑) the box
9. Launching "ground filter"
   - A) Launch [ground_filter] on the Sensing tab by checking (☑) the box
   - B) In the app button, select the corresponding velodyne sensor model
     - ➢ This is used for detection of obstacles over the ground



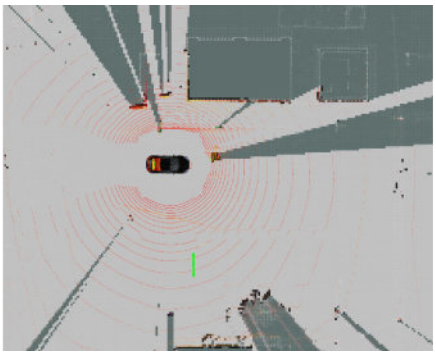
## Path Planning – Step (4/4)

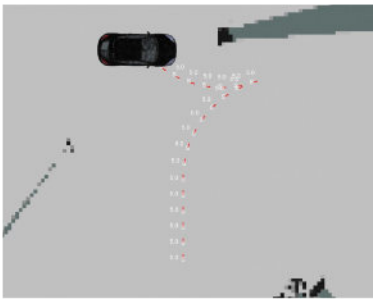
9. Confirm the path in RViz
   - ➢ If light green path（Global Waypoints）and dark green path（Local Waypoints）are displayed, the nodes are working correctly.
   - ➢ If some objects exist in front of the vehicle and object detection success, area boxes will be displayed as shown in the figure below.



## Path Planning for parking: overview

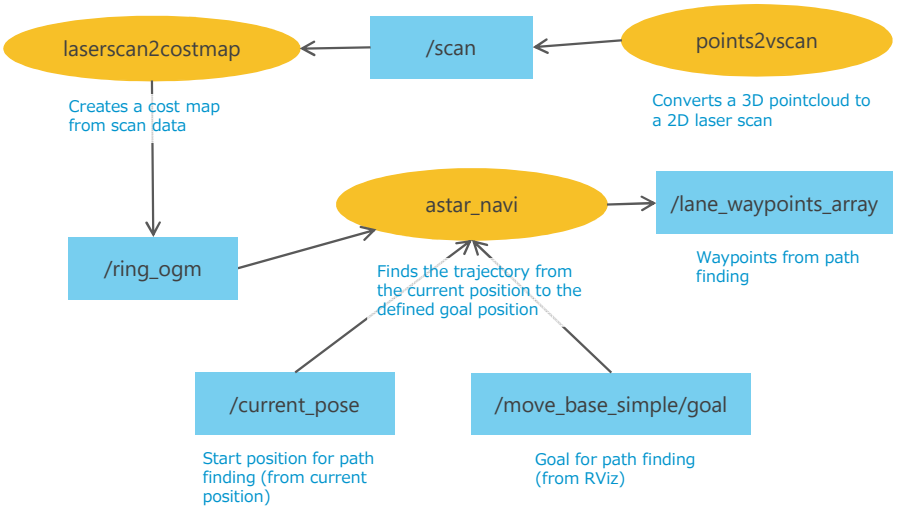
- Create the trajectory(path) from current position to target position
- Uses a cost map and create trajectories considering obstacles
- Target (goal) can be defined in RViz
- The trajectory is created as a set of waypoints





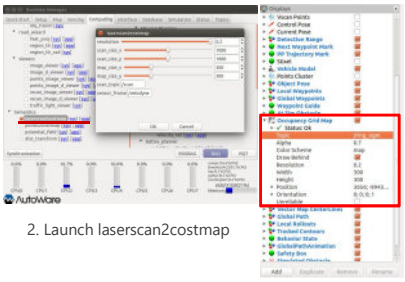


## Path Planning for parking: structure



## Path Planning for parking (1/2)

1. Lauch Virtual Scan



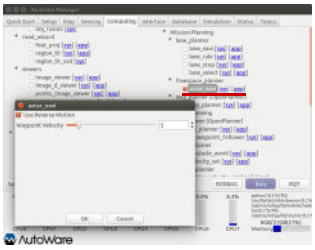
2. Launch laserscan2costmap

3. Display cost map on RViz

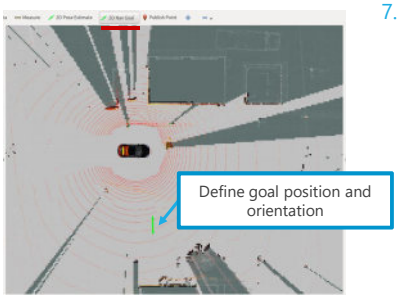
1. Play the ROSBAG and execute localization
   - ➢ Refer to the chapter on localization
2. Launch Virtual Scan
   - A) On [Sensing] tab, click on [Virtual Scan Image] to start it
     - ➢ Velodyne pointcloud is "flattened" to LaserScan
3. Launch laserscan2costmap
   - A) On [Computing] tab, go to [laserscan2costmap] and click on the ☑ button to start it
     - ➢ On the [app] button, you can configure the resolution and map size
4. Display the cost map on RViz
   - A) On the Displays panels of RViz, expand the [Occupancy Grid Map] and use [/ring_ogm] as topic name



## Path Planning for parking (2/2)

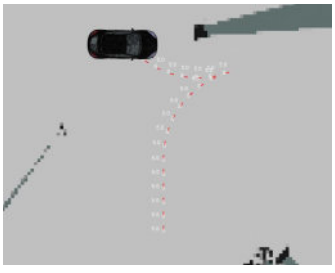


5. Launching astar_navi

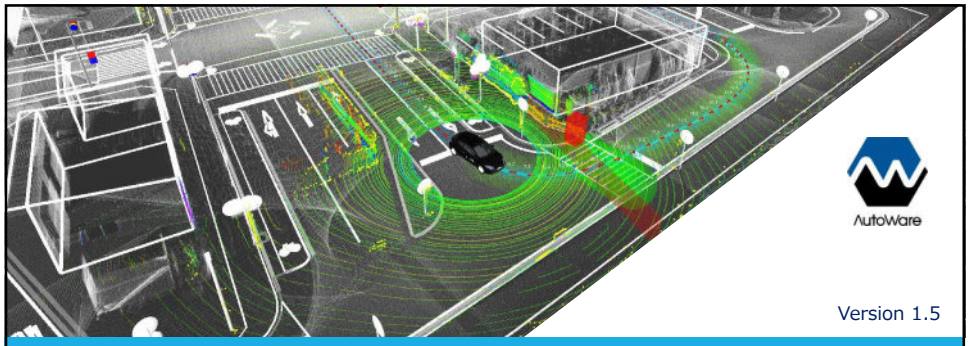



6. Defining goal on RViz

5. Launch astar_navi
   - A) On [Computing] tab, go to [astar_navi] and click on ☑ button to start it
     - ➢ By selecting [Use Reverse Motion], it can generate trajectories with reversing
6. Specify goal position on RViz
   - A) Using the [2D Nav Goal] tool, define the orientation of the goal
     - ➢ On the cost map, gray is free space, black is occupied
     - ➢ Define the goal position on the free space (gray color)
7. Generated trajectory
   - ➢ If path finding succeeds, the waypoints are published and displayed on RViz



7. Generated trajectory



Version 1.5

Autoware Hands-on Experience

# Chapter 6：Path Following and Vehicle Control

## 1. Path following

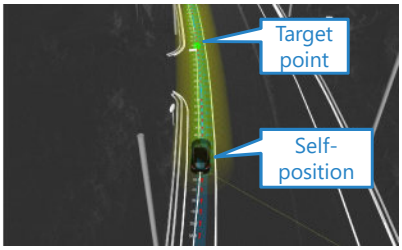


## Path Following – Overview

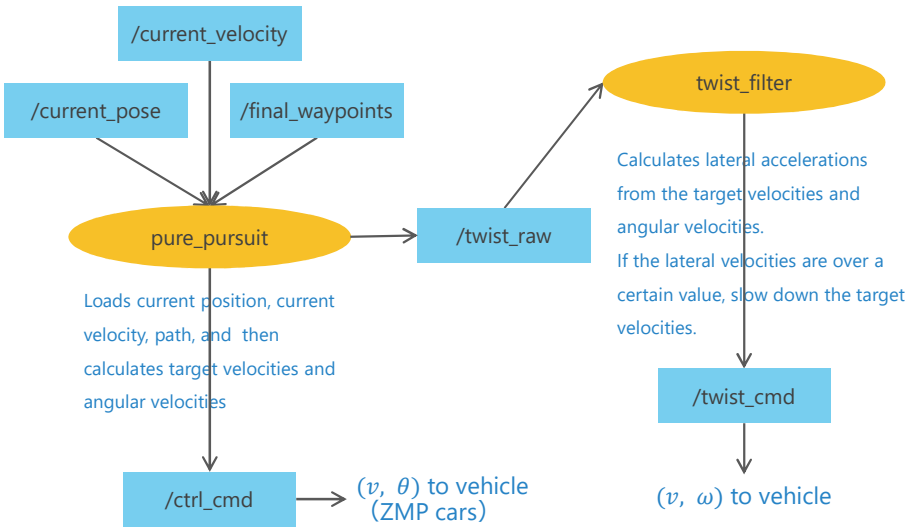
- Calculate vehicle control signals that the vehicle can follow
  - Vehicle control signals $(v,\ \omega)$ or $(v,\ \theta)$ : v...velocity, ω...angular velocity, $\theta$... tire angle
  - Nodes: "pure_pursuit", "twist_filter", "wf_simulator"
  - Subscribed topic: "/current_pose", "/current_velocity", "/final_waypoint"
  - Published topic: "/twist_raw", "/twist_cmd", "/ctrl_cmd"

- Flow of path following and vehicle control

1. "pure_pursuit" node
   - Calculate the curvature of the circle passing through self-position and target points on the path
   - Calculate target angular velocities from the calculated curvature and the current velocity, and then publish them into a topic
2. "twist_filter" node
   - Calculate lateral accelerations from target velocities and angular velocities
   - If the lateral accelerations is over a certain value, slow down the target velocities and publish them into a topic
3. Transmit these nodes to the vehicle for control



## Path Following – Structure

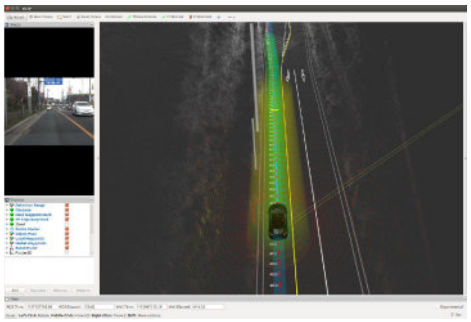





## Path Following – Steps in case of using ROSBAG (1/4)

- Assumptions
  - Localization has been completed, and "/current_pose", "/current_velocity" have been published
  - "/final_waypoints" has been published (path_select is launched)
  - ROSBAG is in Pause

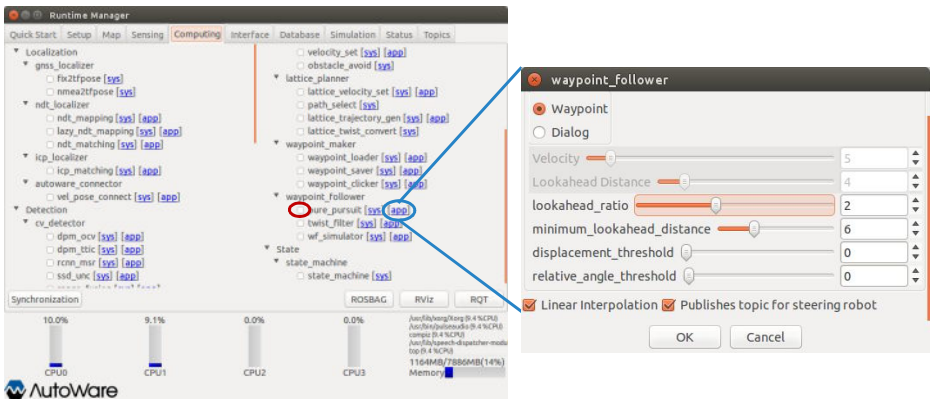
If the above assumptions are completed,
RViz displays somehing as shown below:



## Path Following – Steps in case of using ROSBAG (2/4)

- Specify and launch "pure_pursuit"

1. Clicking [app] on "pure_pursuit", a window will be displayed. Confirm that the values of the parameters are the same as the ones shown in the right-bottom figure
2. Launch pure_pursuit by checking (☑) the box

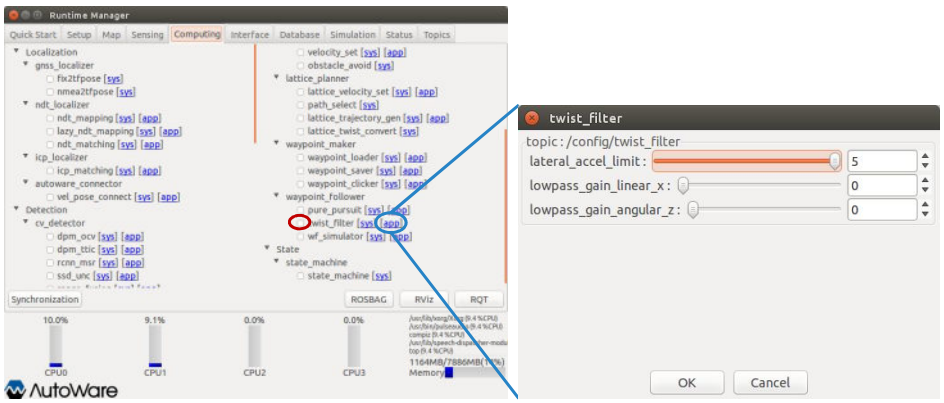





## Path Following – Steps in case of using ROSBAG (3/4)

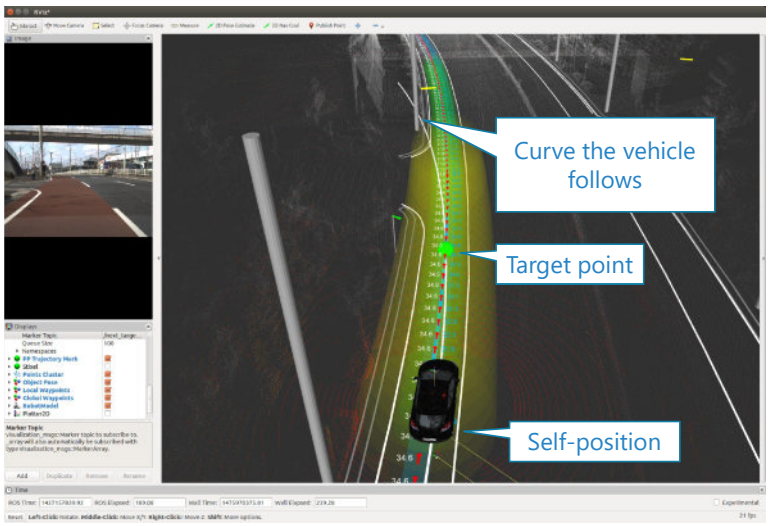
- Specify and launch "twist_filter"

1. Clicking [app] on "twist_filter", a window will be displayed. Confirm that the values of the parameters are the same as the ones shown in the right-bottom figure
2. Launch "twist_filter" by checking (☑) the box



## Path Following – Steps in case of using ROSBAG (4/4)

- Resume playback using [Pause] of ROSBAG, path following results is displayed as shown below



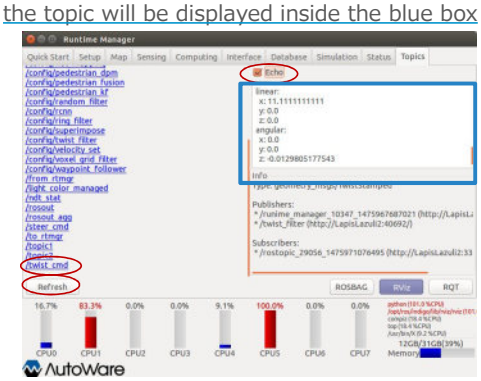



## Path Following and Vehicle Control – Checking topics

- How to check published topics?

1. Check (☑) the box of [Echo] in [Topic] tab
2. Click "/[topic_name]" (here, "/twist_cmd" is clicked and displayed)
   - If the topic doesn't exist, press [Refresh]

After above steps,
the topic will be displayed inside the blue box



## Path Following and Vehicle Control – wf_simulator

- wf_simulator
  - Simulating an ideal self-position and velocity using the received vehicle control signals (v, ω)
  - The problem can be stated as:

$$\begin{cases} x_{i+1} = x_i + v\cos\theta_i\,\Delta t + rnd_x e_{px} \\ y_{i+1} = y_i + v\sin\theta_i\,\Delta t + rnd_y e_{py} \\ \theta_{i+1} = \theta_i + \omega\Delta t + rnd_a e_a \dfrac{\pi}{180} \end{cases}$$

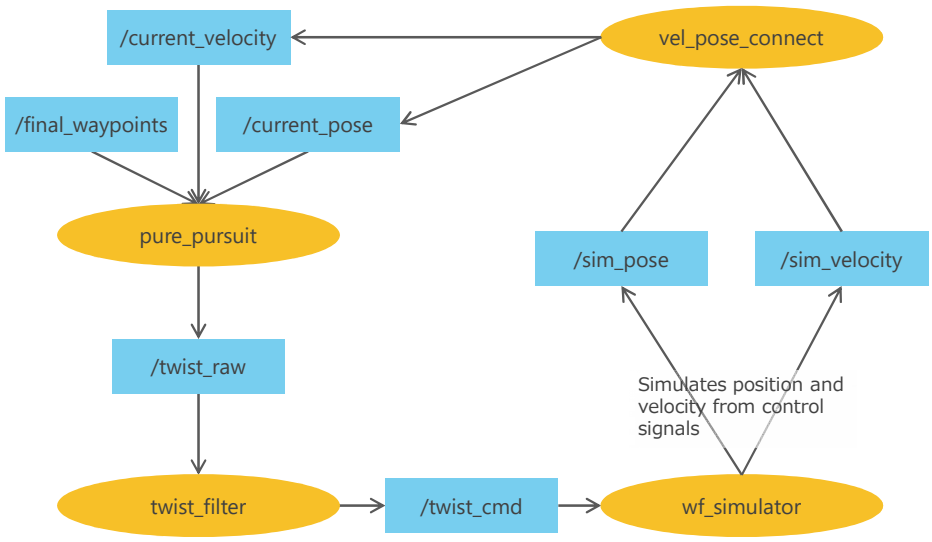
$(x_0, y_0, \theta_0)$ : initial position, $v$ : linear velocity, $\omega$ : angular velocity, $i : 0, 1, \dots, n$

$rnd_x$, $rnd_y$, $rnd_a$ : Random constant $[-1.0 \sim 1.0]$

$e_{px}, e_{py}$ : distance error $[m]$, $e_a$ : angular error $[deg]$

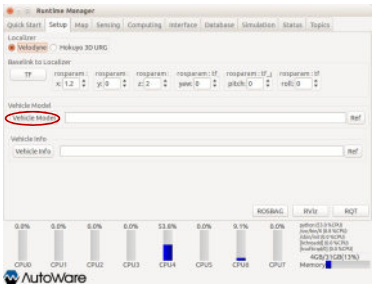


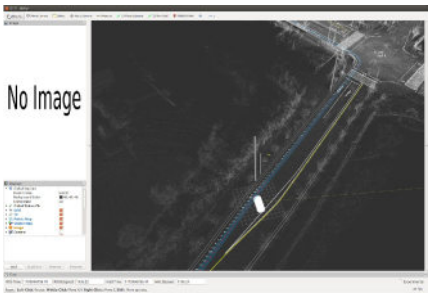
## Path Following and Vehicle Control – wf_simulator



## Path Following and Vehicle Control – wf_simulator (1/8)

- Assumptions
  - Map and "/base_waypoints" have been published ("lane_select" has been launched)
  - If you have run some operations with ROSBAG previously, stop ROSBAG playing and restart TF and RViz
  - Load "/$HOME/Autoware/ros/.config/model/sim_default.urdf" on [Vehicle Model] in [Setup] tab








## Path Following and Vehicle Control – wf_simulator (2/8)

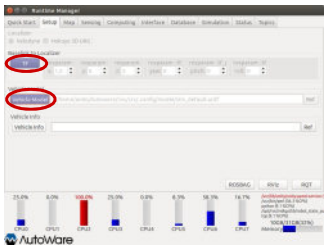
Read map data

※ Make sure rosbag file is stop on simulation tab

1. Read pointcloud map
   - A) On Map tab select the Ref for the point cloud, and select the PCD map file, ex. ~/data/moriyama/pointcloud_map/0.2-bin/*.pcd
   - B) Click on [Point Cloud] button
2. Read the vector map
   - A) On Map tab select the Ref for the vector map, and select the CSV map files, ex. ~/data/moriyama/vector_map/Ver1.20/*.csv
   - B) Click on [Vector Map] button
3. Set the TF for the map
   - A) On Map tab select the Ref for the TF, and select the corresponding launch file, ex. ~/data/moriyama/tf/tf.launch
   - B) Click on [TF] button
     - ➢ The TF between world and map is published

## Path Following and Vehicle Control – wf_simulator (3/8)

Set the TF and vehicle model, load the trajectory



Starting the necessary nodes

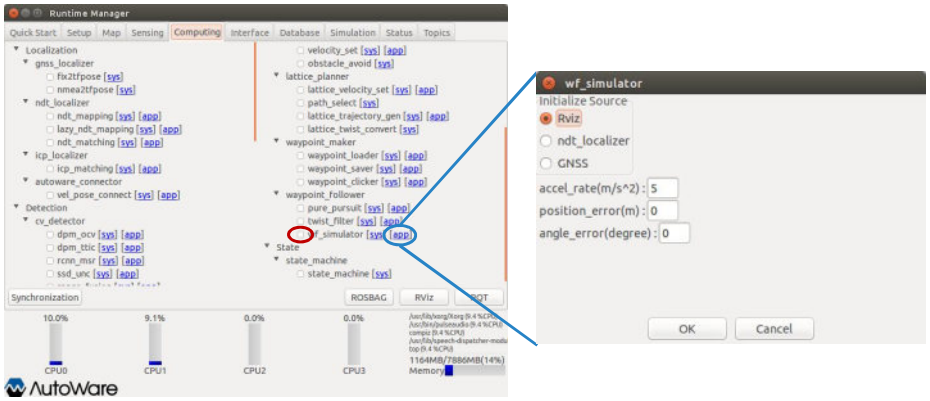
1. Set the TF for the localizer (velodyne) and load vehicle model
   - A) On Setup tab select Velodyne as localizer
   - B) On [Baselink to localizer] define the TF for the velodyne relative to the rear axle (default x:1.2, y:0, z:2.0, yaw:0, pitch:0, roll:0), and click TF
   - C) Load vehicle model
     - ➢ Click on the Ref button of Vehicle model and select the corresponding file, ex. /$HOME/Autoware/ros/src/.config/model/sim_default.urdf
2. To load the trajectory, start the following nodes
   - waypoint_loader
   - lane_rule
   - lane_stop
   - lane_select



## Path Following and Vehicle Control – wf_simulator (4/8)

- "wf_simulator" launching

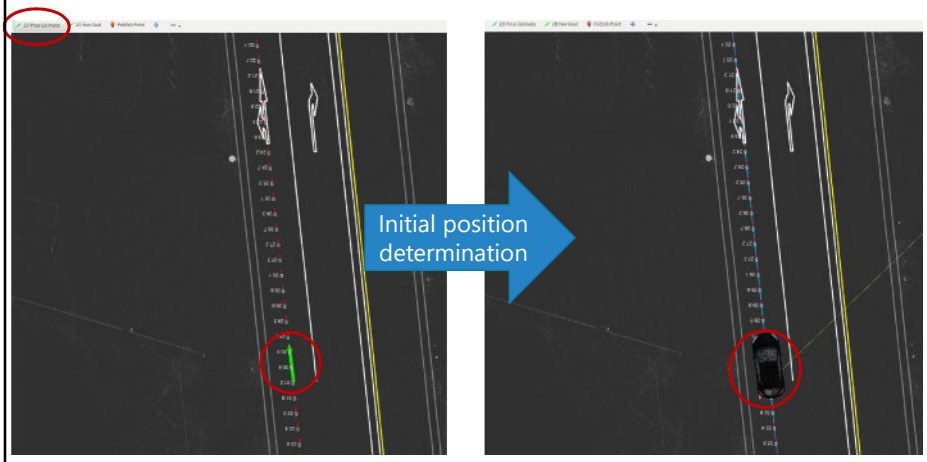
1. Clicking [app] on [wf_simulator], a figure like the one in the bottom-right will be displayed Confirm that the parameters are the same as the ones in the figure
2. Launch [wf_simulator] checking (☑) the box



## Path Following and Vehicle Control – wf_simulator (5/8)

- Launching "wf_simulator"

1. Specify the initial position by 2D Pose Estimate in Rviz, set the Display as "TopDownOrtho"
2. A 3D rendering of vehicle model will be displayed
   If the position of the model is not correct, repeat step 1. again

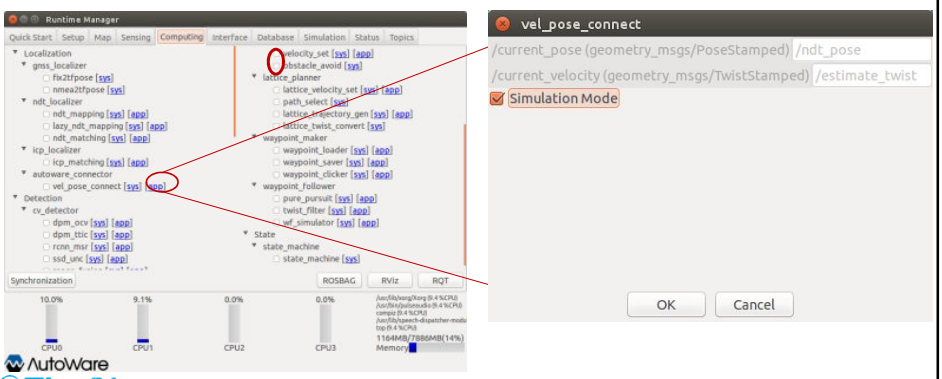




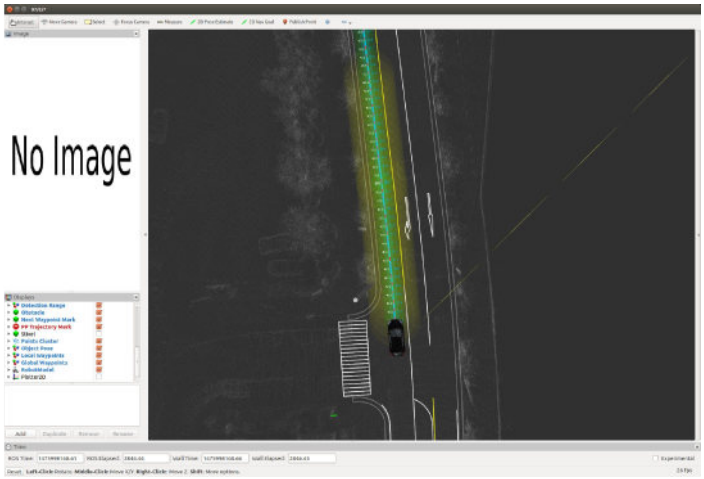

## Path Following and Vehicle Control – wf_simulator (6/8)

- Specifying a Local Path

1. Launch [velocity_set] and [obstacle_avoid] by checking (☑) the boxes
2. Clicking [app] on [vel_pose_connect], the bottom-right figure will be displayed, and then check ☑ the box of Simulation Mode
3. Check (☑) the box of vel_pose_connect



## Path Following and Vehicle Control – wf_simulator (7/8)



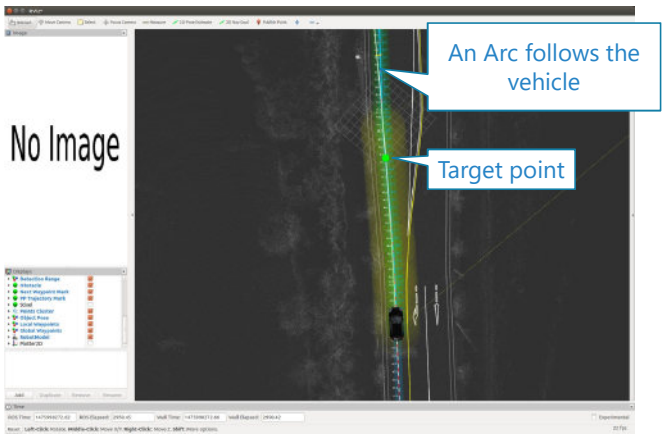

Path is shown as the above figure

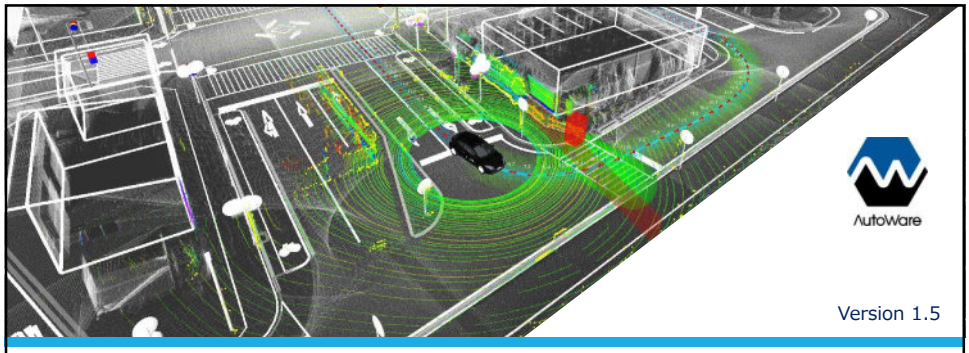


## Path Following and Vehicle Control – wf_simulator (8/8)

- Start path following

1. Launch [pure_pursuit] and [twist_filter]
2. Target points and the arc will be displayed, and then the simulator starts running





Version 1.5

Autoware Hands-on Experience

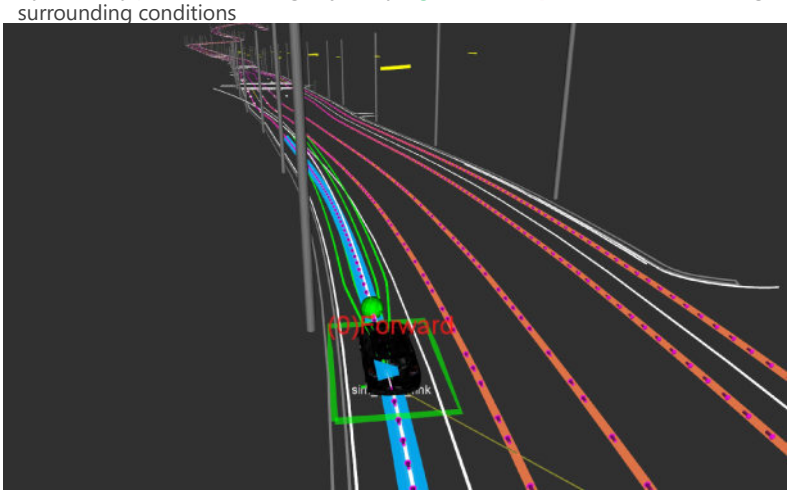
# Chapter 5：Path Following and Vehicle Control

## 2. Open Planner (ADAS planner)



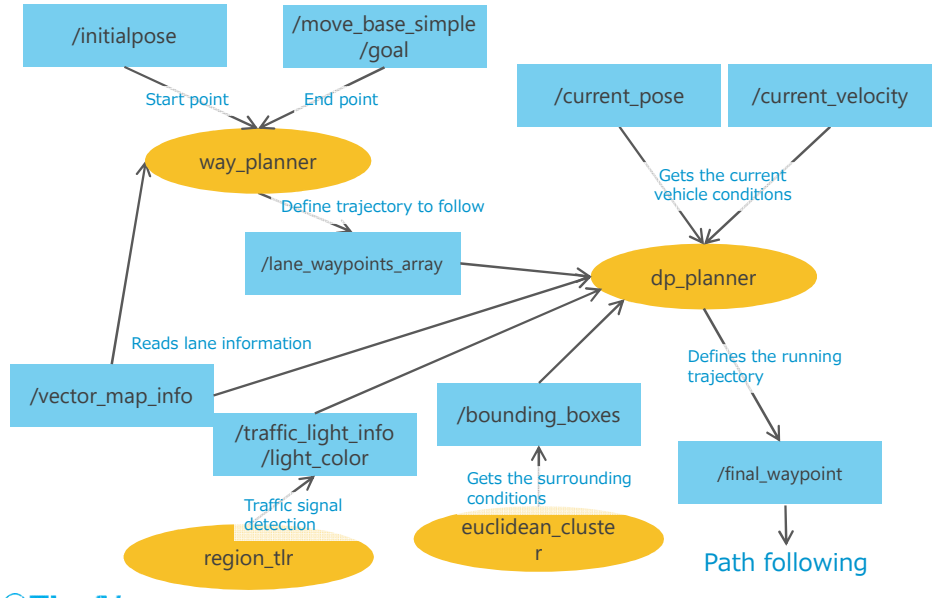
## Open Planner: overview

- Extracts traversable paths from Vector Maps（**orange lines**
- Creates the trajectory to follow（**blue lines**）by defining start and end points
- Dynamically plans the running trajectory（**green lines**, **pink lines**）considering surrounding conditions



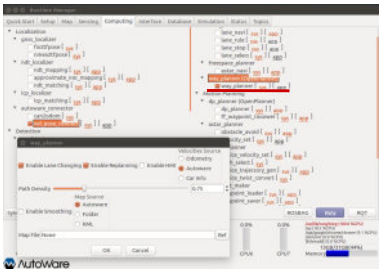
## Open Planner: structure

※Some topics are omitted

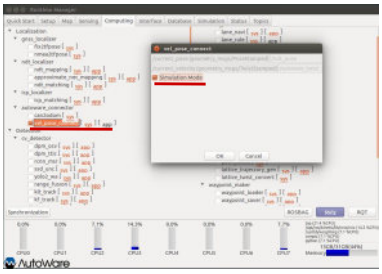





## Open Planner (1/4) – using wf_simulator



2. Launching way_planner



3. Launching vel_pose_connect

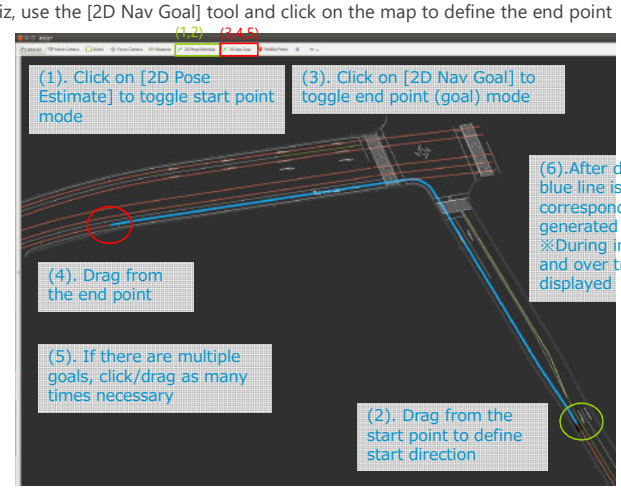
1. Load the map and define the map TF

2. Launch way_planner
   - A) On [Computing] tab go to [way_planner] and click on the ☑ button
     - Before launching, click the [app] button and make sure [Enable Lane Changing] and [Enable Replanning] are checked ☑

3. Launch vel_pose_connect in simulation mode
   - A) On [Computing] tab, go to [vel_pose_connect] and click the ☑ button
     - Before launching, click the [app] button and make sure [Simulation Mode] is checked ☑

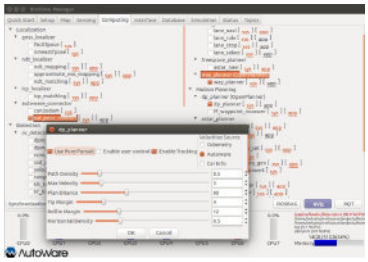
## Open Planner (2/4) – using wf_simulator

4. Configuring start point and end point (goal) for way_planner
   - A) On RViz, use the [2D Pose Estimate] tool and click on the map to define the start point
     - The start direction is defined by dragging
   - B) On RViz, use the [2D Nav Goal] tool and click on the map to define the end point

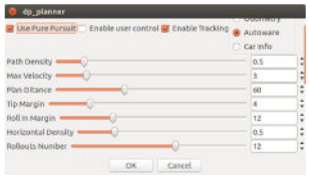





## Open Planner (3/4) – using wf_simulator



5. dp_planner parameters (first half)

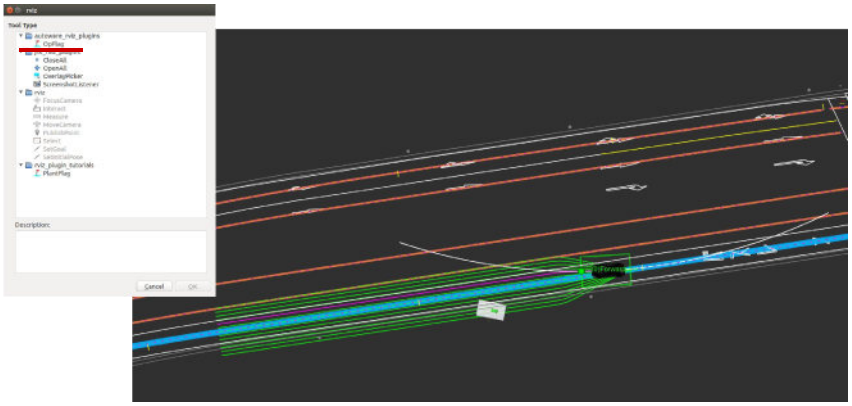


5. dp_planner parameters (second half)

5. Launch dp_planner
   - A) On [Computing] tab go to [dp_planner] and click on the ☑ button
   - B) Some parameters are shown on the left for reference
   - C) By changing the [Roll_Out Number], the number of **green lines** increases, this is the number of possible paths to calculate increase
6. Launch Pure-Pursuit
   - A) On [Computing] tab go to [pure_pursuit] and click on the ☑ button
7. Launch Twist_filter
   - A) On [Computing] tab go to [twist_filter] and click on the ☑ button

## Open Planner (4/4) – using wf_simulator

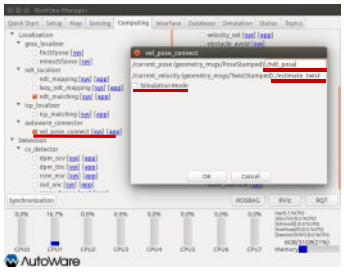
8. On RViz define the starting position, check results
   A) Using the [+] button on the RViz toolbar, select [OpFlag] to add virtual obstacles
   B) Use [2D Estimate Pose] tool one more time on the vehicle starting position
   C) Verify that there are several **green lines** (local waypoints) going from the vehicle position towards the end point, and the vehicle avoids obstacles



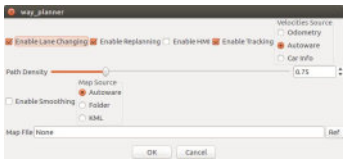
8. Defining start point and verifying results in RViz



## Open Planner (1/3) – using rosbag



1. Launching vel_pose_connect



2. Launching way_planner

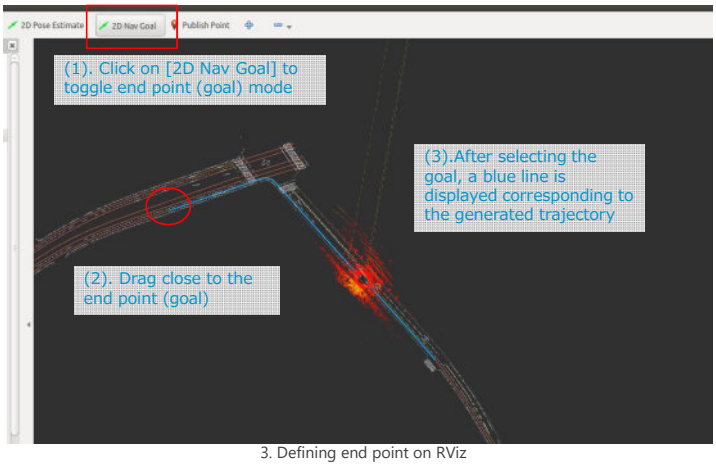
1. Play the ROSBAG and execute localization
   - Refer to the chapter on localization

2. Launch vel_pose_connect
   A) On [Computing] tab go to [vel_pose_connect] and click on the ☑ button
   - Click on [app] button and verify [/ndt_pose] and [/estimate_twist] are checked, and [Simulation Mode] is NOT checked

3. Launch way_planner
   A) On [Computing] tab go to [way_planner] and click on the ☑ button
   - Before launching, click the [app] button and make sure [Enable Lane Changing] and [Enable Replanning] are checked ☑

## Open Planner (2/3) – using rosbag

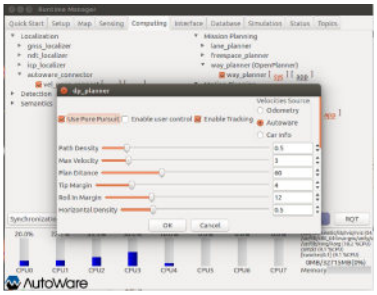
4. Define the end point (goal) for way_planner on RViz
   A) (optional) Play the rosbag until vehicle is localized and over the lane
      - On the Moriyama ROSBAG this is about after 85(s)
   B) On RViz, use the [2D Nav Goal] tool and define the end point (goal) on the map




3. Defining end point on RViz

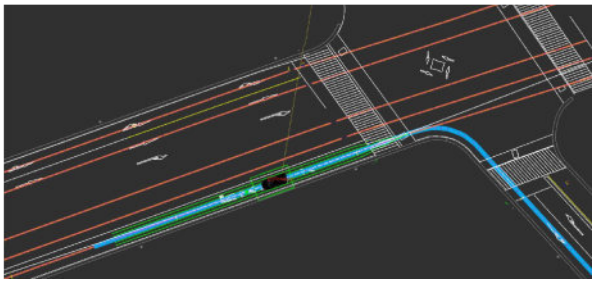


## Open Planner (3/3) – using rosbag




4 . Launching dp_planner (parameters)

5. Launch dp_planner
   A) On [Computing] tab go to [dp_planner] and click on the ☑ button
6. (optional) Launch Euclidean Cluster
   A) On [Computing] tab go to [euclidean_cluster] and click on the ☑ button
      - By launching Euclidean cluster, obstacle avoidance is activated
7. Verify on RViz
   A) Verify that there are several **green lines** (local waypoints) going from the vehicle position towards the end point, and the vehicle avoids obstacles
   B) Check if some of the rollouts have a different color, it means dp_planner is working correctly



5. Verifying on RViz

Tier IV
Intelligent Vehicle
www.tier4.jp

133