

Part II. Autoware Install / Build / Run (2 hours)

숭실대학교 ICHTHUS 팀
김강희 교수(khkim@ssu.ac.kr)
이효은, 최규진, 백한나 연구원

Outline

- ❖ Installation overview
- ❖ Docker-based installation
- ❖ Docker image build
- ❖ Installation from scratch (can be skipped if CUDA is not used)
- ❖ Source build (can be skipped since it is part of docker image build)
- ❖ Simulation in Autoware
- ❖ rosbag play

Software Stack

docker image: lab_kinetic_cuda.tar.gz
or lab_kinetic.tar.gz

Application: **Autoware 1.12** **CUDA runtime 10**

Middleware: **ROS kinetic**

System Libraries: **Ubuntu 16.04**

Virtualization: **Docker** **NVIDIA runtime**

Host OS: **Ubuntu 16.04** **CUDA driver 10**

Installation Overview

<https://gitlab.com/autwarefoundation/autware.ai/autware/wikis/Installation>

❖ Autoware 설치 방법

● host 환경에 설치하는 방법

- ❖ 하나의 ROS 버전과 Autoware 버전을 사용하는 경우에 추천함
- ❖ 변경 사항들이 스토리지에 바로 저장됨

● **docker 환경에 설치하는 방법 < 이번 실습에서 사용하는 방법**

- ❖ 여러 ROS 버전과 Autoware 버전을 사용하는 경우에 추천함
 - Autoware 실행 이미지를 여러 개 관리할 때 필수적으로 사용함
- ❖ **변경 사항들이 docker image에 저장되지 않음!**
 - docker container는 하나의 filesystem image 상에서 실행되며, container 안에서 변경된 파일들은 container 종료시 image에 저장되지 않음
 - 따라서 변경된 파일들을 유지하기 위해서는 container를 종료하기 전에 docker image를 새로 생성(commit)해야 함
- ❖ host의 cuda driver와 cuda runtime toolkit을 모두 사용할 수 있음
- ❖ host와 container 사이에 /home/autoware/shared_dir 폴더를 공유하여 두 시스템 사이에 파일 공유가 가능함

Source Build Requirements

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/Source-Build>

❖ 실습 목표: ubuntu 16.04 & ros kinetic & docker & autoware 1.12

Autoware Version	Ubuntu 14.04	Ubuntu 16.04	Ubuntu 18.04
v1.12.0		X	X
v1.11.1		X	
v1.11.0		X	
v1.10.0		X	
v1.9.1	X	X	
v1.9.0	X	X	

Product	Ubuntu 14.04	Ubuntu 16.04	Ubuntu 18.04
ROS	Indigo	Kinetic	Melodic
Qt	4.8.6 or higher	5.2.1 or higher	5.9.5 or higher
CUDA (optional)	8.0GA(?)	9.0	10.0

Note: autoware 1.12를 cuda와 함께 빌드하기 위해서는 반드시 cuda v10.0 을 사용해야 함. cuda v10.1을 사용하면 빌드 실패함

Computer Spec Recommendation

<https://gitlab.com/autwarefoundation/autware.ai/autware/wikis/Docker>

- ❖ Generic amd64(64-bit x86)
 - Ubuntu 16.04 / 18.04
 - CPU
 - ❖ Intel Core i7 (preferred)
 - ❖ Intel Core i5
 - ❖ AMD Ryzen 7 (preferred)
 - ❖ AMD Ryzen 5
 - 16GB ~ 32GB of RAM
 - > 30GB of Storage (SSD preferred)
 - NVIDIA GTX GeForce GPU (>= 980M)
- ❖ NVIDIA Drive
 - Drive PX2, Drive Xavier
 - > 30 GB SSD

이번 실습에서 사용하는 랩탑 사양

- ❖ Generic amd64 (64-bit x86)
 - Ubuntu 16.04 LTS
 - CPU
 - ❖ Intel Core i7-4700HQ @ 2.4GHz (quadcore)
 - 16GB RAM
 - 250GB SSD
 - no NVIDIA GTX GeForce GPU
→ GPU 없이, 즉 cuda 없이 autware 빌드!

Docker-based Installation

Docker-based Installation Steps

Note 1: 이번 실습에서는 ros kinetic & autoware source files & prebuilt autoware binaries를 담고 있는 docker image를 제공받아 사용함

Note 2: 이번 실습에서 사용하는 docker image는 cuda를 사용하지 않으므로 step 3은 필요 없음. 또한 step 4에서 run.sh 에서 CUDA="off"하고 docker image를 download해야 함

- ❖ [host] step 1: create a user named 'autoware'
- ❖ [host] step 2: install Docker CE
- ❖ [host] step 3: install NVIDIA Docker Runtime
- ❖ [host] step 4: run an Autoware Docker image

Step 1: Create a user named 'autoware'

- ❖ create a user named 'autoware'
 - \$ sudo adduser autoware
 - ❖ host에서 실행하는 위 명령이 UID = 1000 이라는 것을 보장하지는 않음 (docker container 안에서 autoware UID = 1000 임)
 - ❖ 만약 user rubicom이 UID = 1000이고, user autoware가 UID = 1001이라면, 아래 결과를 얻도록 관련 파일들을 편집할 것
- ❖ edit /etc/passwd
 - rubicom:x:1001:1001:Rubicom,,,:/home/rubicom:/bin/bash
 - autoware:x:1000:1000:Autoware,,,:/home/autoware:/bin/bash
- ❖ edit /etc/group
 - sudo:x:27:autoware # add 'autoware' into group 'sudo'
 - rubicom:x:1001 # previously 1000
 - autoware:x:1000 # previously 1001
- ❖ change the ownership of files owned by 'autoware'
 - \$ sudo chown -R 1000:1000 /home/autoware
 - \$ sudo chown -R 1001:1001 /home/rubicom
- ❖ edit /etc/gdm3/custom.conf
 - AutomaticLogin=autoware

Step 2: Install Docker CE

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/docker-installation>

❖ Old Docker Cleanup

- \$ sudo apt-get remove docker docker-engine docker.io

❖ Docker CE Installation

- \$ sudo apt-get update # update package lists
- \$ sudo apt-get install apt-transport-https ca-certificates curl software-properties-common # permit apt-get to access the repository, which uses HTTPS
- \$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - # add Docker's official GPG key
- \$ sudo apt-key fingerprint 0EBFCD88 # validate the key

```
pub 4096R/0EBFCD88 2017-02-22
Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid Docker Release (CE deb)
sub 4096R/F273FCD8 2017-02-22
```

- \$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu \$(lsb_release -cs) stable"
 - # add the repository to your system

Step 2: Install Docker CE

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/docker-installation>

❖ Docker CE Installation (cont'd)

- `$ sudo apt-get update # update package lists with the new repository`
- `$ sudo apt-get install docker-ce # install the latest version of Docker CE`
- `$ sudo docker run hello-world # validate the installation succeeded`

Step 3: Install NVIDIA Docker Runtime (GPU 사용시)

[https://github.com/NVIDIA/nvidia-docker/wiki/Installation-\(version-2.0\)](https://github.com/NVIDIA/nvidia-docker/wiki/Installation-(version-2.0))

- ❖ Prerequisites for running nvidia-docker 2.0
 - GNU/Linux x86_64 with kernel version > 3.10
 - Docker >= 1.12
 - NVIDIA GPU with Architecture > Fermi (2.1)
 - NVIDIA drivers ~= 361.93 (untested on older versions)
- ❖ Remove nvidia-docker 1.0
 - `$ docker volume ls -q -f driver=nvidia-docker | xargs -r -l{} -n1 docker ps -q -a -f volume={} | xargs -r docker rm -f`
 - `$ sudo apt-get purge nvidia-docker`

Step 3: Install NVIDIA Docker Runtime (GPU 사용시)

[https://github.com/NVIDIA/nvidia-docker/wiki/Installation-\(version-2.0\)](https://github.com/NVIDIA/nvidia-docker/wiki/Installation-(version-2.0))

- ❖ Set up the repository configuration (assuming Debian-based distributions)
 - `$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -`
 - `$ distribution=$(. /etc/os-release;echo IDVERSION_ID)`
 - `$ curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list`
 - `$ sudo apt-get update`
- ❖ Updating repository keys
 - `$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -`
- ❖ Install the nvidia-docker2 package and reload the Docker daemon configuration:
 - `$ sudo apt-get install nvidia-docker2`
 - `$ sudo pkill -SIGHUP dockerd`
- ❖ Basic usage
 - `$ docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi`

Step 4: Run an Autoware Docker Image

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/Generic-x86-Docker>

Note on file permissions (both in host or in container)

- `$ cp -r target_dir ~/shared_dir`
- `$ cd ~/shared_dir && sudo chown -R $(id -u):$(id -g) *`

Using Pre-built Autoware Docker Images

❖ docker image 안에 Autoware release를 build하는데 필요한 모든 요구사항들이 이미 image 안에 설치되어 있으며, 해당 release도 이미 build된 상태로 제공됨 → /home/autoware/Autoware

- `$ git clone https://gitlab.com/autowarefoundation/autoware.ai/docker.git`
- `$ cd docker/generic`
- `$ sudo ./run.sh -r kinetic -s # autoware/autoware:latest-kinetic-cuda 설치`

autoware@rubicom-MS-7B09:~/docker/generic\$ sudo docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
autoware/autoware	local-melodic-cuda	9461c01555ed	4 hours ago	8.46GB
autoware/autoware	local-melodic-base-cuda	205560ac8d8c	4 hours ago	5.4GB
autoware/autoware	local-melodic-base	a7defe6bfad0	5 hours ago	3.17GB
ros	melodic	8c7e1b93c802	2 days ago	1.25GB
autoware/autoware	latest-melodic-cuda	3ff5da2f0f72	2 weeks ago	8.45GB
autoware/autoware	latest-kinetic-cuda	22a5ed3276d6	2 weeks ago	8.07GB
nvidia/cuda	latest	010a71dc59db	4 weeks ago	2.81GB
ubuntu	latest	4c108a37151f	6 weeks ago	64.2MB
nvidia/opengl	1.0-glvnd-runtime-ubuntu18.04	f7d2b59439b6	2 months ago	315MB
autoware/autoware	1.10.0-kinetic-cuda	7926d37a198a	6 months ago	7.98GB
hello-world	latest	fce289e99eb9	7 months ago	1.84kB

Step 4: Run an Autoware Docker Image

<https://gitlab.com/autwarefoundation/autware.ai/autware/wikis/Generic-x86-Docker>

```
#!/bin/bash
# Default settings
CUDA="on"
IMAGE_NAME="autware/autware"
TAG_PREFIX="latest"
ROS_DISTRO="kinetic"
BASE_ONLY="false"
PRE_RELEASE="off"
AUTOWARE_HOST_DIR=""
USER_ID="$(id -u)"
...
RUNTIME="--runtime=nvidia"
SUFFIX="-cuda"
...
IMAGE=$IMAGE_NAME:$TAG_PREFIX-$ROS_DISTRO$SUFFIX
docker run -it --rm $VOLUMES W
  --env="XAUTHORITY=${XAUTH}" W
  --env="DISPLAY=${DISPLAY}" W
  --env="USER_ID=$USER_ID" W
  --privileged W
  --net=host W
  $RUNTIME W
  $IMAGE
```

→ 사용자가 원하대로 편집할 것

Docker Image Build (covering 'Installation from Scratch' and 'Source Build')

One shot one kill : build.sh

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/Generic-x86-Docker>

Creating a Custom Autoware Docker Image

- ❖ image build 전에 build.sh 파일을 원하는대로 편집함 → 다음 페이지 참조
 - \$ git clone
https://gitlab.com/autowarefoundation/autoware.ai/docker.git
 - \$ cd docker/generic
 - \$ sudo ./build.sh # 'local-' prefix로 시작하는 docker image 생성함
 - \$ sudo docker images
 - ❖ 최종 이미지 이름은 **autoware/autoware:local-kinetic-cuda** 임

One shot one kill : build.sh

```
#!/bin/bash
# Default settings
CUDA="on"
IMAGE_NAME="autoware/autoware"
TAG_PREFIX="local"
ROS_DISTRO="kinetic"
BASE_ONLY="false"
VERSION="master"
...
BASE=$IMAGE_NAME:$TAG_PREFIX-$ROS_DISTRO-base
CUDA_SUFFIX="-cuda"
DOCKERFILE="Dockerfile"
...
docker build W
  --rm W
  --tag $IMAGE_NAME:$TAG_PREFIX-$ROS_DISTRO$CUDA_SUFFIX W
  --build-arg FROM_ARG=$BASE$CUDA_SUFFIX W
  --build-arg ROS_DISTRO=$ROS_DISTRO W
  --build-arg VERSION=$VERSION W
  --file $DOCKERFILE .
```

→ 사용자가 원하대로 편집할 것

Installation from Scratch

Installation Steps from Scratch

Note 1: 이번 실습에서는 ros kinetic & autoware source files & prebuilt autoware binaries를 담고 있는 docker image를 제공받아 사용함

Note 2: 다음 절차는 host 또는 docker 환경에서 autoware build & run 환경을 새로 구성하려는 경우에 필요함

Note 3: cuda driver는 host 환경에서 설치되어야 하며, cuda runtime는 autoware 실행 환경과 동일한 환경(host 또는 docker 환경)에서 설치되어야 함 → 즉 step 3 중 cuda driver 설치 과정은 docker 환경에서 필요 없음

- ❖ step 1: install ROS Kinetic
- ❖ step 2: install system dependencies for Ubuntu 16.04 / Kinetic
- ❖ step 3: install CUDA 10.0 (GPU 사용시)
 - cuda driver는 반드시 host 환경에서 설치되어야 함

Step 1: Install ROS Kinetic

<http://wiki.ros.org/kinetic/Installation/Ubuntu>

- ❖ Set up your sources.list
 - `$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`
- ❖ Set up your keys
 - `$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654`
- ❖ Install
 - `$ sudo apt update; sudo apt install ros-kinetic-desktop-full`
- ❖ Initialize rosdep
 - `$ sudo rosdep init; rosdep update`
- ❖ Set up environment
 - `$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc; source ~/.bashrc`
 - ❖ Note: docker container 안에서는 /etc/profile.d/ros.sh 가 /opt/ros/kinetic/setup.bash 을 자동으로 실행시킴
- ❖ Set up tools for building packages
 - `$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential`

Step 2: Install system dependencies for Ubuntu 16.04 / Kinetic

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/Source-Build>

- ❖ \$ sudo apt-get update
- ❖ \$ sudo apt-get install -y python-catkin-pkg python-rosdep ros-kinetic-catkin gksu
- ❖ \$ sudo apt-get install -y python3-pip python3-colcon-common-extensions python3-setuptools python3-vcstool
- ❖ \$ pip3 install -U setuptools

Step 3: Install CUDA 10.0 (GPU 사용시)

<https://docs.nvidia.com/cuda/archive/10.0/cuda-installation-guide-linux/index.html>

- ❖ Verify you have a cuda-capable gpu
 - \$ lspci | grep -i nvidia
- ❖ Install the kernel headers
 - \$ sudo apt-get install linux-headers-\$(uname -r)
- ❖ Download cuda-10.0 package from <https://developer.nvidia.com/cuda-10.0-download-archive> # 상기 URL에는 없는 절차임
- ❖ Install the package using Package Manager
 - \$ sudo dpkg -i cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb
 - \$ sudo apt-key add /var/cuda-repo-10-0-local-10.0.130-410.48/7fa2af80.pub
 - \$ sudo apt-get update
 - \$ sudo apt-get install cuda-10-0 # 반드시 cuda-10-0이라고 명시할 것! 상기 URL을 따라 'sudo apt-get install cuda' 명령을 실행하면 cuda-10-1이 설치됨
- ❖ Set up environment variables in ~/.bashrc
 - \$ echo "export PATH=/usr/local/cuda-10.0/bin:\${PATH}" >> ~/.bashrc
 - \$ echo "export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:\${LD_LIBRARY_PATH}" >> ~/.bashrc
 - \$ source ~/.bashrc

Step 3: Install CUDA 10.0 (GPU 사용시)

❖ cuda driver 설치 확인: nvidia-smi ❖ cuda runtime 설치 확인: deviceQuery

```
autoware@rubicom-MS-7B09:~/docker/generic$ /usr/local/cuda/extras/demo_suite/deviceQuery
/usr/local/cuda/extras/demo_suite/deviceQuery Starting...
```

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1080 Ti"

CUDA Driver Version / Runtime Version	10.1 / 10.0
CUDA Capability Major/Minor version number:	6.1
Total amount of global memory:	11175 MBytes (11718230016 bytes)
(28) Multiprocessors, (128) CUDA Cores/MP:	3584 CUDA Cores
GPU Max Clock rate:	1683 MHz (1.68 GHz)
Memory Clock rate:	5505 Mhz
Memory Bus Width:	352-bit
L2 Cache Size:	2883584 bytes
Maximum Texture Dimension Size (x,y,z)	1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
Maximum Layered 1D Texture Size, (num) layers	1D=(32768), 2048 layers
Maximum Layered 2D Texture Size, (num) layers	2D=(32768, 32768), 2048 layers
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block:	65536
Warp size:	32
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
Max dimension size of a thread block (x,y,z):	(1024, 1024, 64)
Max dimension size of a grid size (x,y,z):	(2147483647, 65535, 65535)
Maximum memory pitch:	2147483647 bytes
Texture alignment:	512 bytes
Concurrent copy and kernel execution:	Yes with 2 copy engine(s)
Run time limit on kernels:	Yes
Integrated GPU sharing Host Memory:	No
Support host page-locked memory mapping:	Yes
Alignment requirement for Surfaces:	Yes
Device has ECC support:	Disabled
Device supports Unified Addressing (UVA):	Yes
Device supports Compute Preemption:	Yes
Supports Cooperative Kernel Launch:	Yes
Supports MultiDevice Co-op Kernel Launch:	Yes
Device PCI Domain ID / Bus ID / location ID:	0 / 65 / 0
Compute Mode:	

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.1, CUDA Runtime Version = 10.0, NumDevs = 1, Device0 = GeForce GTX 1080 Ti
Result = PASS

cuda driver version 10.1이더라도 cuda runtime version 10.0 이면 autoware 1.12를 빌드하는데 문제 없음

Source Build

Source Build Steps

Note 1: 이번 실습에서는 ros kinetic & autoware source files & prebuilt autoware binaries를 담고 있는 docker image를 제공받아 사용함

Note 2: 다음 절차는 host 또는 docker 환경에서 autoware build & run 하려는 경우에 필요함

- ❖ step 1: Create a workspace
- ❖ step 2: Download the workspace configuration for autoware.ai
- ❖ step 3: Download the source files into the workspace
- ❖ step 4: Install dependencies using rosdep
- ❖ step 5: Compile the workspace

Source Build Steps

<https://gitlab.com/autwarefoundation/autware.ai/autware/wikis/Source-Build>

❖ step 1: Create a workspace

- \$ mkdir -p Autware/src && cd Autware

❖ docker image 안에서는 autware.ai 아닌 Autware라는 폴더명을 사용할 것. 왜냐하면 /home/autware/.bashrc가 /home/autware/Autware/install/local_setup.bash 스크립트를 찾기 때문임

❖ step 2: Download the workspace configuration for autware.ai

- \$ wget -O autware.ai.repos
"https://gitlab.com/autwarefoundation/autware.ai/autware/raw/1.12.0/autware.ai.repos?inline=false" # for the 1.12.0 release
- \$ wget -O autware.ai.repos
"https://gitlab.com/autwarefoundation/autware.ai/autware/raw/master/autware.ai.repos?inline=false" # for the bleeding edge version

Source Build Steps

<https://gitlab.com/autwarefoundation/autware.ai/autware/wikis/Source-Build>

- ❖ step 3: Download the source files into the workspace
 - `$ vcs import src < autoware.ai.repos`
- ❖ step 4: Install dependencies using rosdep
 - `$ rosdep update`
 - `$ rosdep install -y --from-paths src --ignore-src --rosdistro kinetic`
- ❖ step 5: Compile the workspace
 - `$ AUTOWARE_COMPILE_WITH_CUDA=1 colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release # with cuda support`
 - `$ colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release # without cuda support`

Note : SSD (single shot detection)와 같은 DNN 기반 노드들은 자동적으로 빌드되지 않음. 해당 노드의 README 파일을 참조할 것

Source Build Steps

<https://gitlab.com/autowarefoundation/autoware.ai/autoware/wikis/Source-Build>

❖ build success screenshot

```
Finished <<< waypoint_planner [44.0s]
Starting >>> lattice_planner
Finished <<< lane_planner [52.3s]
Starting >>> waypoint_maker
Finished <<< dp_planner [53.0s]
Finished <<< lidar_localizer [3min 2s]
Finished <<< op_global_planner [37.0s]
Finished <<< op_utilities [37.8s]
Finished <<< lidar_kf_contour_track [39.3s]
Finished <<< op_local_planner [40.4s]
Finished <<< lattice_planner [32.1s]
Finished <<< waypoint_maker [26.8s]
Finished <<< op_simulation_package [56.2s]
```

Summary: 141 packages finished [5min 25s]

build success!!

47 packages had stderr output: adi_driver astar_search autoware_camera_lidar_calibrator autoware_connector autoware_driveworks_gmsl_interface autoware_driveworks_interface autoware_pointgrey_drivers citysim data_preprocessor decision_maker dp_planner glviewer kitti_player kvaser lidar_apollo_cnn_seg_detect lidar_euclidean_cluster_detect lidar_localizer lidar_point_pillars lidar_shape_estimation map_file microstrain_driver mqtt_socket ndt_cpu ndt_gpu object_map op_ros_helpers op_simulation_package op_utilities pcl_omp_registration pixel_cloud_fusion points_downsampler points_preprocessor qpoases_vendor range_vision_fusion road_occupancy_processor sick_ldmrs_tools sick_lms5xx trafficlight_recognizer vector_map_server vision_darknet_detect vision_segment_enet_detect vision_sd_detect vlg22c_cam way_planner waypoint_follower waypoint_maker waypoint_planner

Simulation in Autoware

Step 1: Prepare shared_dir

- ❖ 제공된 USB DISK를 usb slot에 꽂은 후에 다음 절차를 진행함
 - \$ cp /media/autoware/AUTOWARE/shared_dir.tar.gz ~/
 - \$ cd # change to home directory
 - \$ tar xvzf shared_dir.tar.gz # unpack files into ~/shared_dir folder
 - \$ cd shared_dir
 - \$ ls -al # only files in blue are used in this simulation
 - ❖ default.rviz # rviz default configuration file
 - ❖ EgoCar.csv # open_planner's saved initialpose and goalpose
 - ❖ kcity_map/ # contains map files for kcity
 - tf.launch # initial transform to make the map visible in rviz
 - kcity_20190629_0.2_update.pcd # point map (pcd format)
 - kcity_avc2019_final/[1-9]/*.csv # vector map (csv format)
 - ❖ rosbag_demo/ # contains rosbag & launch files for rosbag play
 - ❖ src/ # contains modified autoware source files for this demo

Step 2: Run 'local-kinetic' container

❖ [host] \$ cd ~/docker/generic

❖ [host] \$ sudo ./run.sh -s

'local-kinetic' or 'local-kinetic-cuda' 이미지 기반 container 실행

Step 3: Prepare runtime_manager

- ❖ [container] \$ cd Autoware
- ❖ [container] \$ find . -name run -print
 - ./src/autoware/utilities/runtime_manager/scripts/run
 - ./install/runtime_manager/share/runtime_manager/scripts/run
- ❖ [container] \$ edit the above two files as follows
 - add the following command "cp ~/shared_dir/default.rviz ~/.rviz/" right before "roslaunch runtime_manager runtime_manager_dialog.py"

```
# boot runtime_manager
${TERMINAL} ${RUNMGR_DISPLAY_OPTION} ${OPTION_WORKING_DIR}=${MY_PATH} ${OPTION_COMMAND}="bash -c 'source ../../../../../../setup.bash; mkdir ~/.rviz; cp `rospack find autoware_quickstart_examples`/launch/rosbag_demo/default.rviz ~/.rviz/; cp ~/shared_dir/default.rviz ~/.rviz/; roslaunch runtime_manager runtime_manager_dialog.py'
```

Note: run script 편집 후에 현재 container 의 변경 사항들을 저장하는 docker image를 docker commit 명령으로 생성할 것을 권장함

Step 4: Launch runtime_manager

- ❖ [container] \$ cd Autoware
- ❖ [container] \$ roslaunch runtime_manager runtime_manager.launch

```
autoware@rubicom-MS-7B09:~/Autoware$ roslaunch runtime_manager runtime_manager.launch
```

```
... logging to /home/autoware/.ros/log/7c39d67c-b803-11e9-96
rubicom-MS-7B09-1735.log
Checking log directory for disk usage. T
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage
```

```
started roslaunch server http://rubicom-
```

```
SUMMARY
```

```
=====
```

```
PARAMETERS
```

```
* /rostdistro: melodic
* /rosversion: 1.14.3
```

```
NODES
```

```
/
  run (runtime_manager/run)
```

```
auto-starting new master
```

```
process[master]: started with pid [1745]
ROS_MASTER_URI=http://localhost:11311
```

```
setting /run_id to 7c39d67c-b803-11e9-96
```

```
process[rosout-1]: started with pid [175
```

```
started core service [/rosout]
```

```
process[run-2]: started with pid [1762]
```

```
[run-2] process has finished cleanly
```

```
log file: /home/autoware/.ros/log/7c39d6
```

Runtime Manager

Quick Start Setup Map Sensing Computing Interface Database

Map Ref

Sensing Ref

Localization Ref

Detection Ref

Mission Planning Ref

Motion Planning Ref

Android Tablet Oculus Rift Vehicle Gateway Remote Control Cloud Data

Auto Pilot ROSBAG RViz RQT

0.0%

CPU0CPU1CPU2CPU3CPU4CPU5CPU6CPU7CPU8CPU9CPU10CPU11CPU12CPU13CPU14CPU15CPU16CPU17

Step 4-1: Load map and tf files

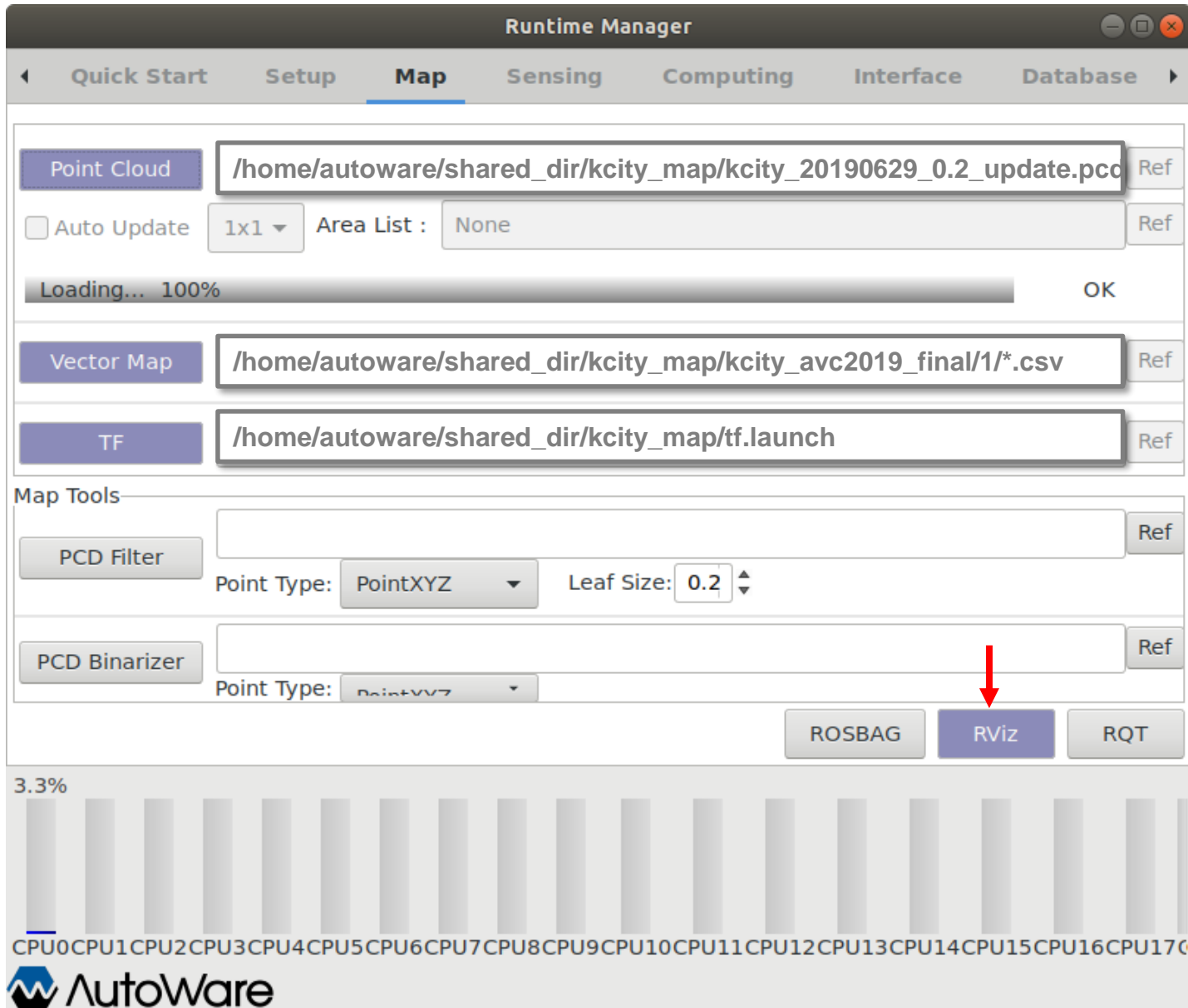
The screenshot shows the 'Runtime Manager' window with the 'Map' tab selected. The 'Map' tab contains several configuration sections:

- Point Cloud:** A red box highlights the file path `/home/autoware/shared_dir/kcity_map/kcity_20190629_0.2_update.pcd`. A red arrow labeled (4) points to the 'Point Cloud' button, and another red arrow labeled (1) points to the 'Ref' button.
- Vector Map:** A red box highlights the file path `/home/autoware/shared_dir/kcity_map/kcity_avc2019_final/1/*.csv`. A red arrow labeled (5) points to the 'Vector Map' button, and another red arrow labeled (2) points to the 'Ref' button.
- TF:** A red box highlights the file path `/home/autoware/shared_dir/kcity_map/tf.launch`. A red arrow labeled (6) points to the 'TF' button, and another red arrow labeled (3) points to the 'Ref' button.

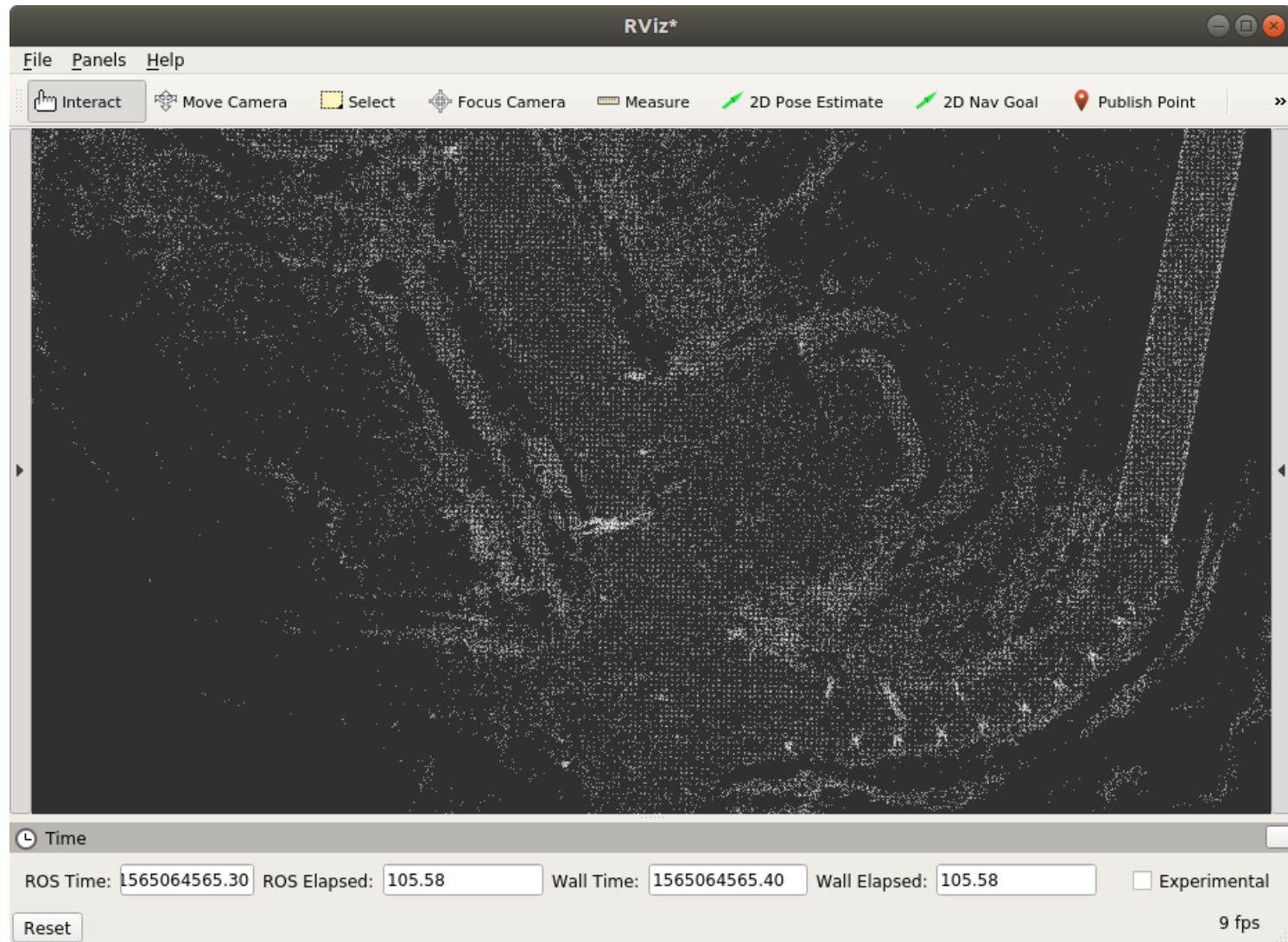
Below these sections is the 'Map Tools' section, which includes 'PCD Filter' and 'PCD Binarizer' buttons, each with a 'Ref' button. The 'PCD Filter' section also has a 'Point Type' dropdown set to 'PointXYZ' and a 'Leaf Size' input set to '0.2'. The 'PCD Binarizer' section has a 'Point Type' dropdown set to 'PointXYZ'.

At the bottom of the window, there is a CPU usage bar chart showing 0.0% usage across 17 CPUs (CPU0 to CPU17). The Autoware logo is visible at the bottom left.

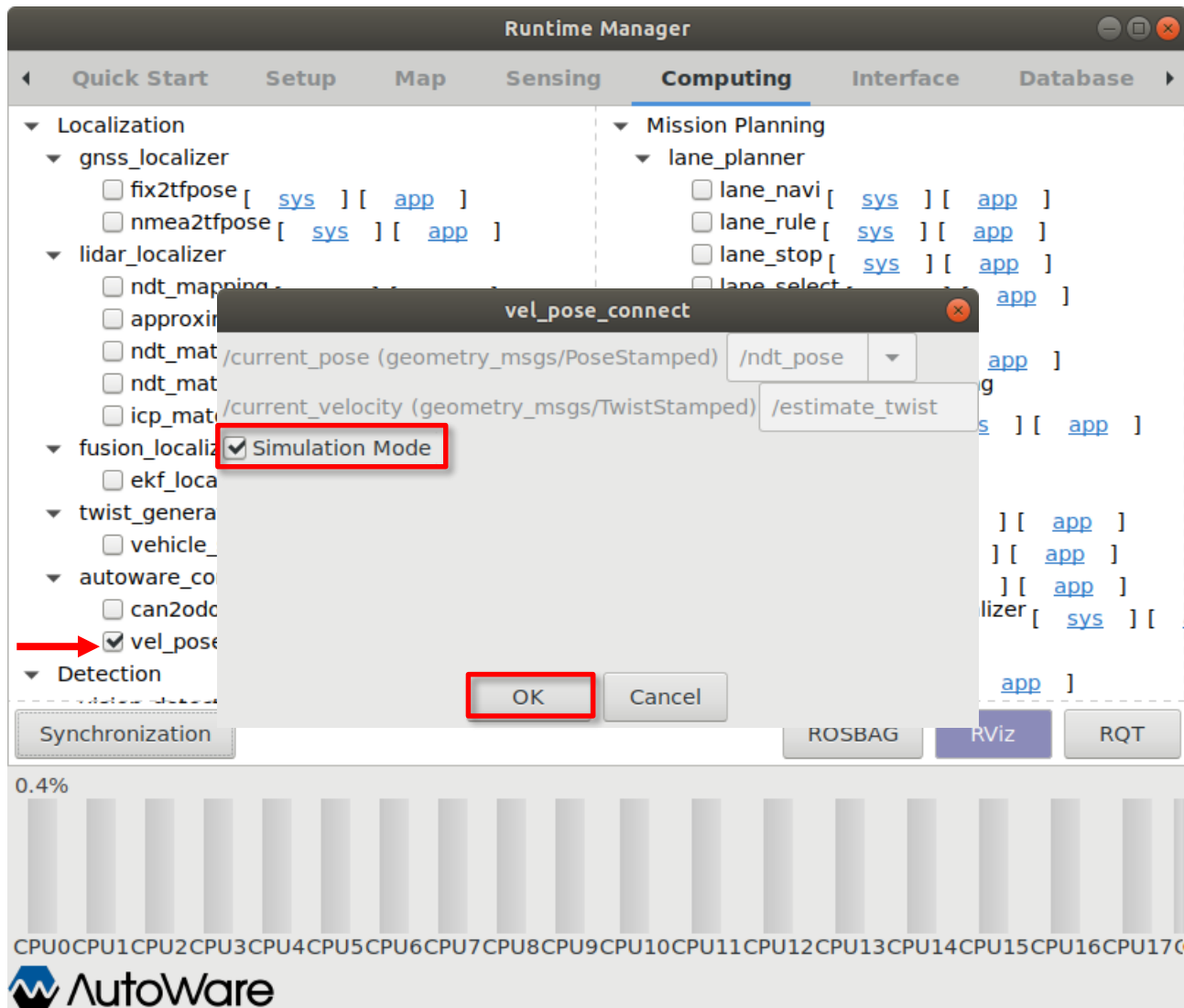
Step 4-2: Launch rviz



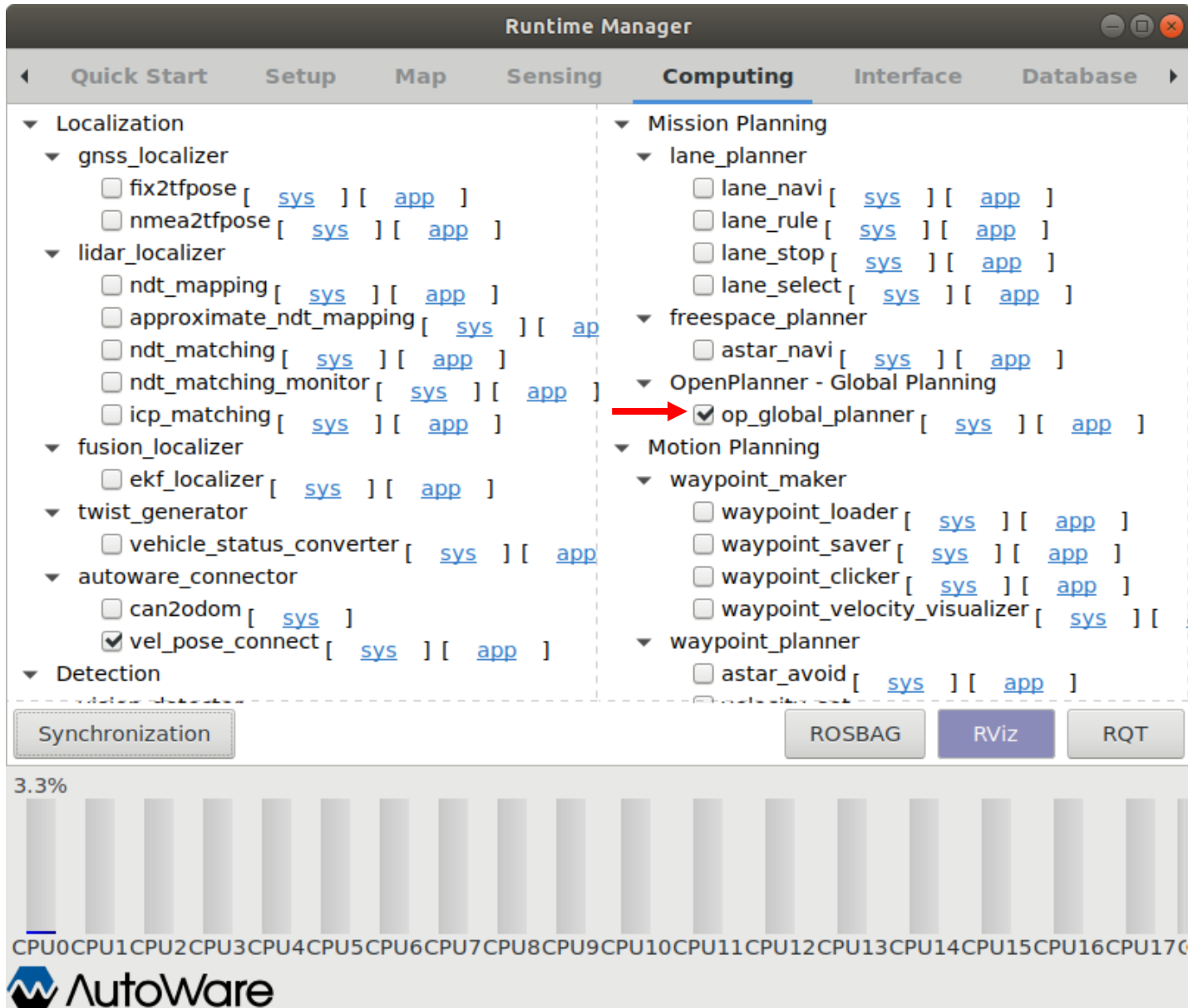
Step 4-2: Launch rviz



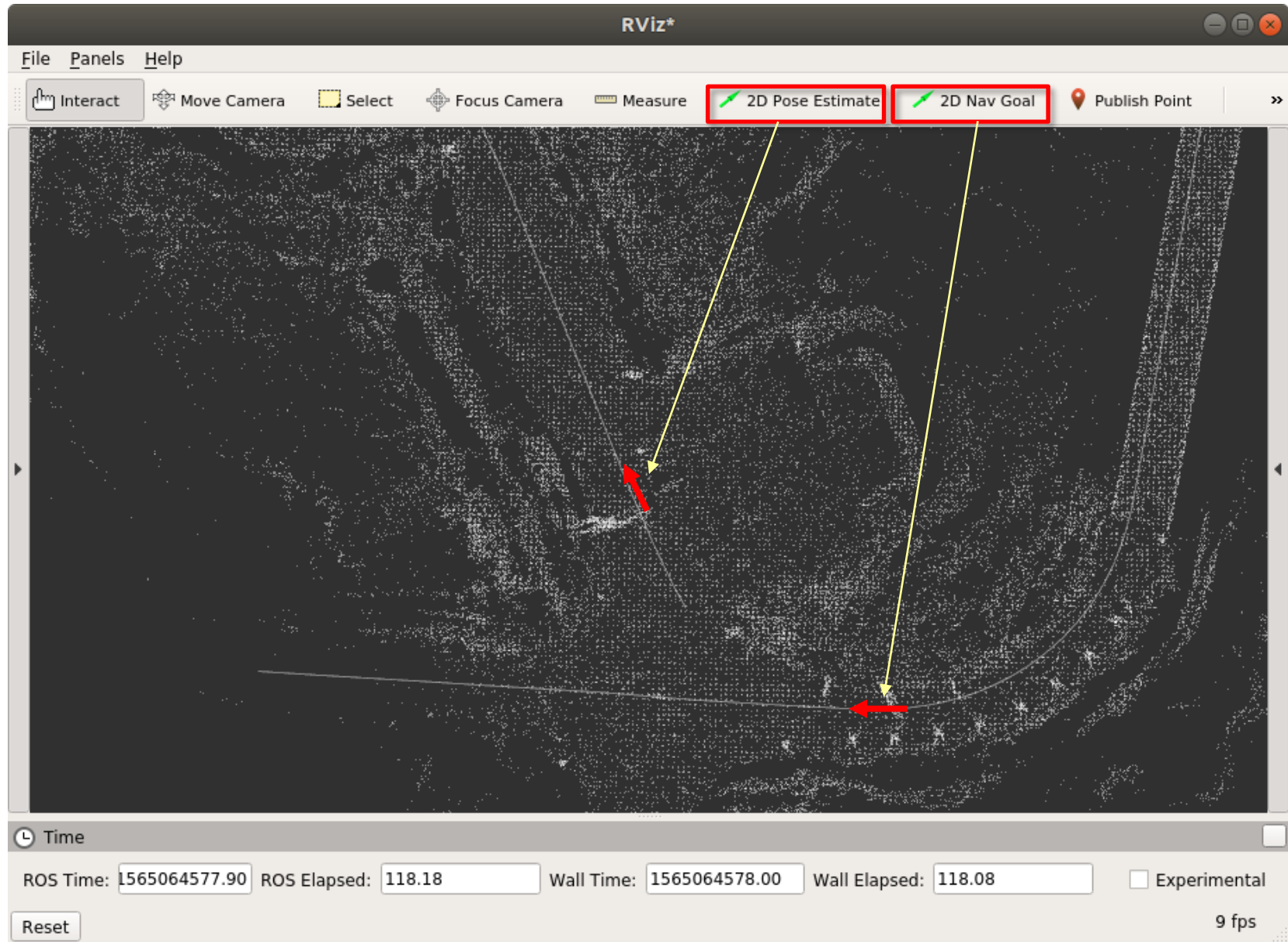
Step 4-3: Launch vel_pose_connect



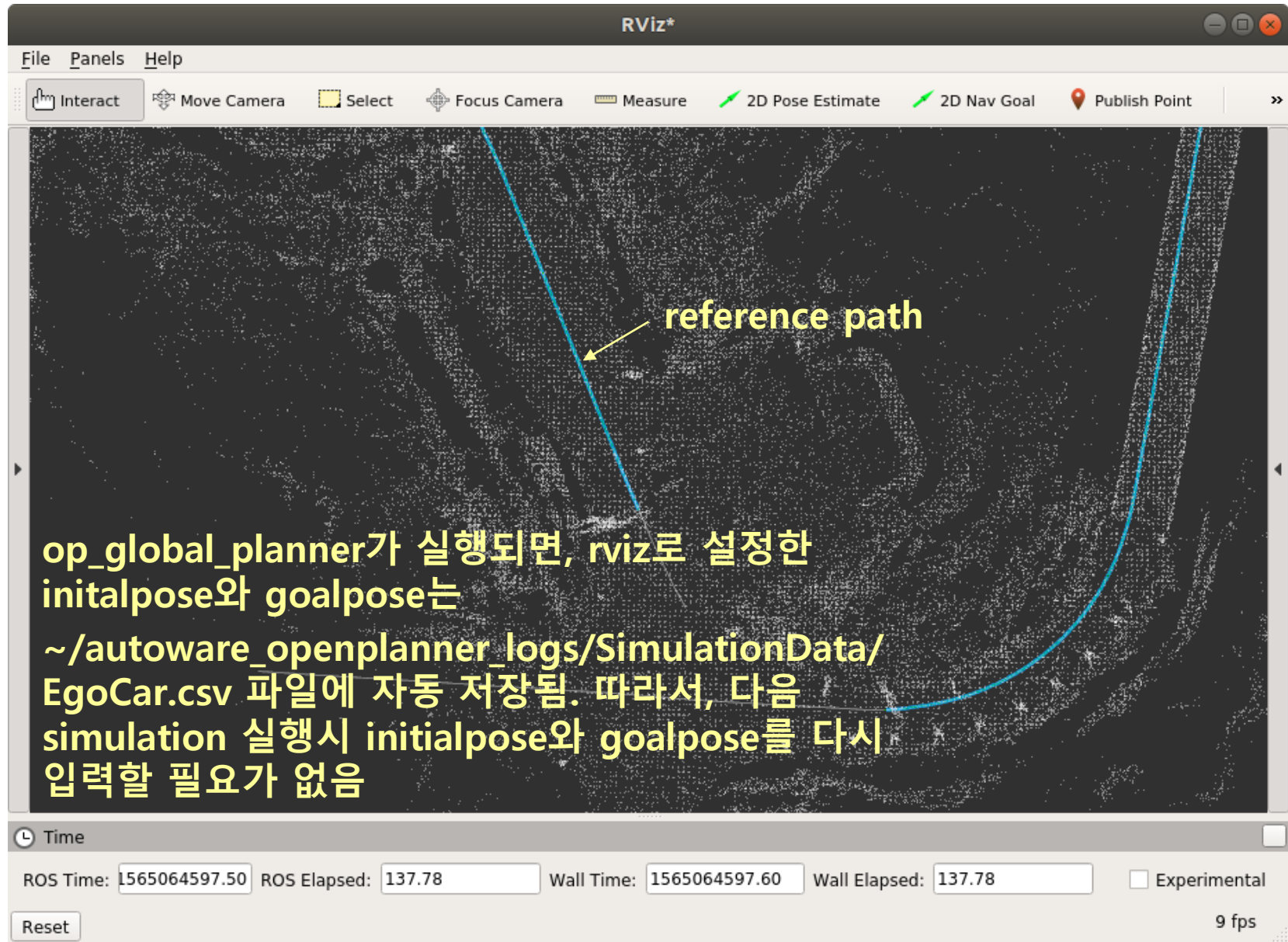
Step 4-4: Launch op_global_planner



Step 4-5: Set start and goal poses



Step 4-5: Set start and goal poses



Step 4-6: Launch wf_simulator & OpenPlanner

Runtime Manager

Quick Start Setup Map Sensing **Computing** Interface Data

Localization

- gnss_localizer**
 - ☐ fix2tfpose [sys] [app]
 - ☐ nmea2tfpose [sys] [app]
- lidar_localizer**
 - ☐ ndt_mapping [sys] [app]
 - ☐ approximate_ndt_mapping [sys] [app]
 - ☐ ndt_matching [sys] [app]
 - ☐ ndt_matching_monitor [sys] [app]
 - ☐ icp_matching [sys] [app]
- fusion_localizer**
 - ☐ ekf_localizer [sys] [app]
- twist_generator**
 - ☐ vehicle_status_converter [sys] [app]
- autoware_connector**
 - ☐ can2odom [sys]
 - ☒ vel_pose_connect [sys] [app]
- Detection**

Computing

- ☐ waypoint_velocity_visualizer [sys] [app]
- waypoint_planner**
 - ☐ astar_avoid [sys] [app]
 - ☐ velocity_set [sys] [app]
- waypoint_follower**
 - ☐ pure_pursuit [sys] [app]
 - ☐ mpc_follower [sys] [app]
 - ☐ twist_filter [sys] [app]
 - ☒ wf_simulator [sys] [app]
- OpenPlanner - Local planning**
 - ☒ op_common_params [sys] [app]
 - ☒ op_trajectory_generator [sys] [app]
 - ☐ op_motion_predictor [sys] [app]
 - ☒ op_trajectory_evaluator [sys] [app]
 - ☒ op_behavior_selector [sys] [app]
- OpenPlanner - Utilities**
 - ☐ op_pose2tf [sys] [app]
 - ☐ op_bag_player [sys] [app]

Configuration Dialog: wf_simulator

Initialize Source: ☒ RVIZ ☐ NDT ☐ GNSS ☐ ORIGIN

☐ add_measurement_noise

☐ use_waypoints_for_z_position_source

vehicle_model_type Source: IDEAL_TWIST

tf simulation_frame_id: **sim_base_link**

tf lidar_frame_id: velodyne

output pose topic: current_pose

output velocity topic: current_velocity

velocity limit [m/s]: 30

angular velocity limit [rad/s]: 3

acceleration limit [m/ss]: 1

angular acceleration limit [rad/ss]: 1

velocity time delay [s]: 0.25

velocity time constant [s]: 0.6

angular velocity time delay [s]: 0.2

angular velocity time constant [s]: 0.4

steering angle limit [rad]: 1

steering angular velocity limit [rad/s]: 0.5

steering time delay [s]: 0.24

steering time constant [s]: 0.27

OK Cancel

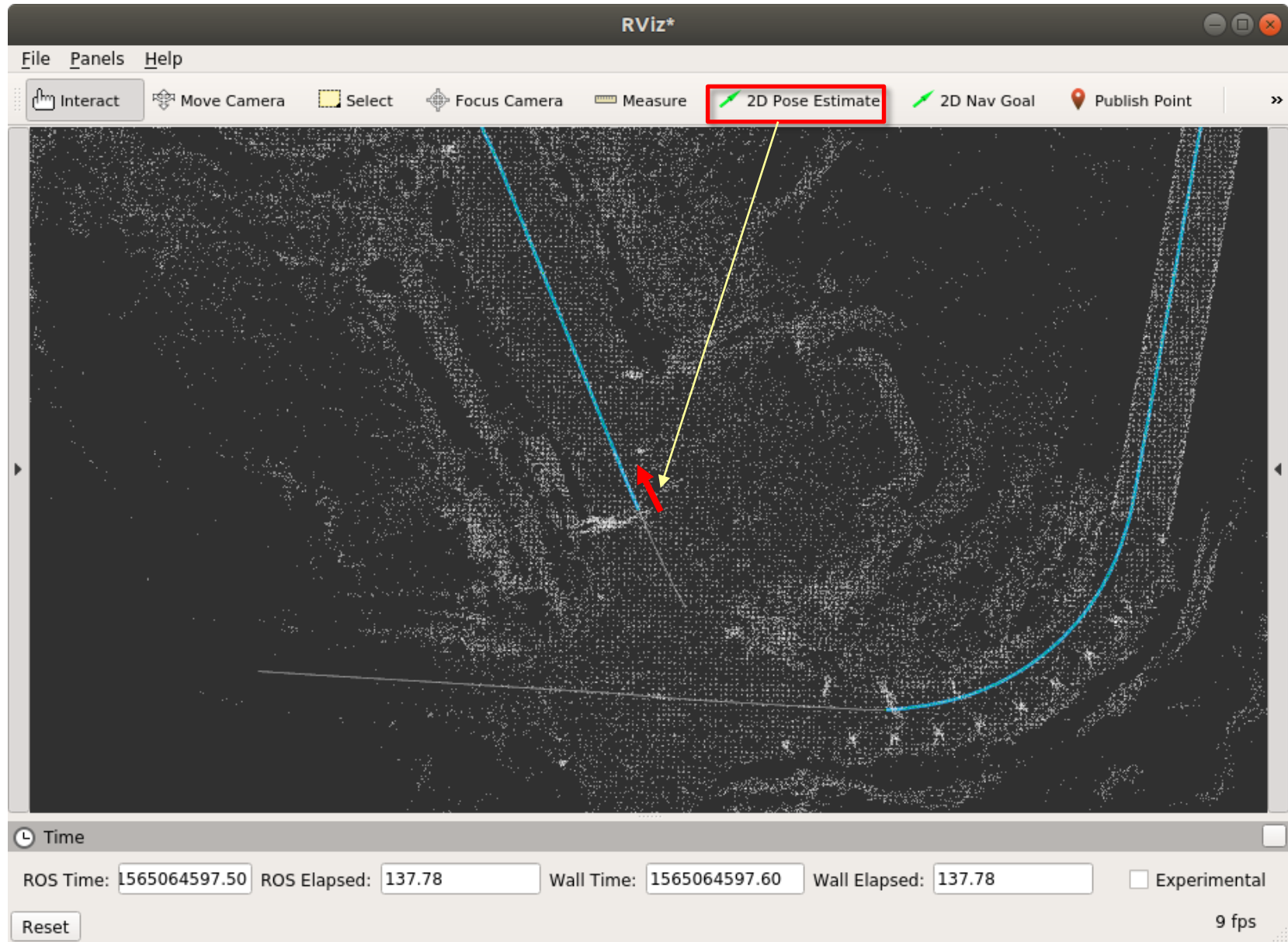
Performance Monitor

2.1%

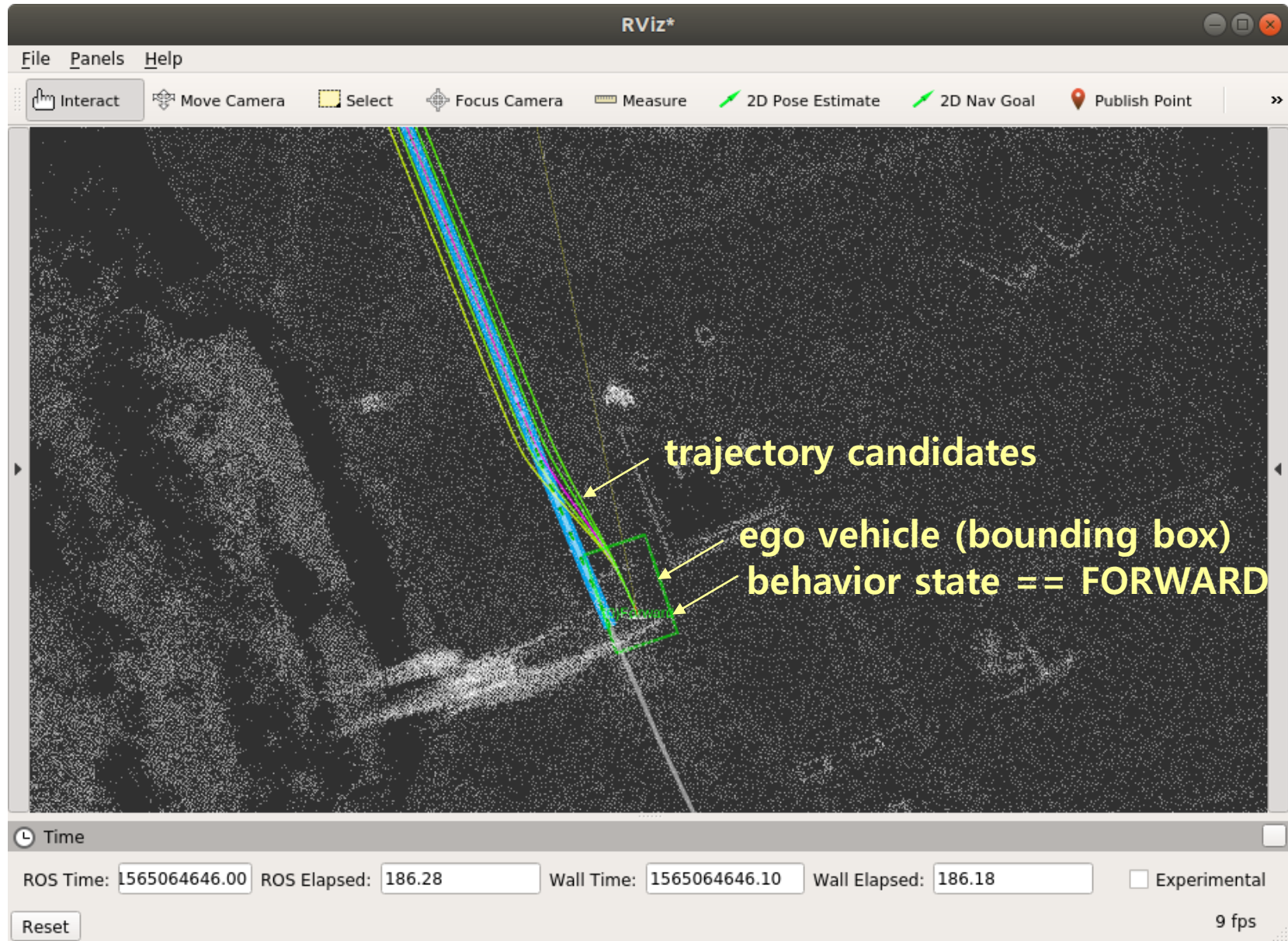
CPU0 CPU1 CPU2 CPU3 CPU4 CPU5 CPU6 CPU7 CPU8 CPU9 CPU10 CPU11 CPU12 CPU13 CPU14 CPU15 CPU16 CPU17

AutoWare

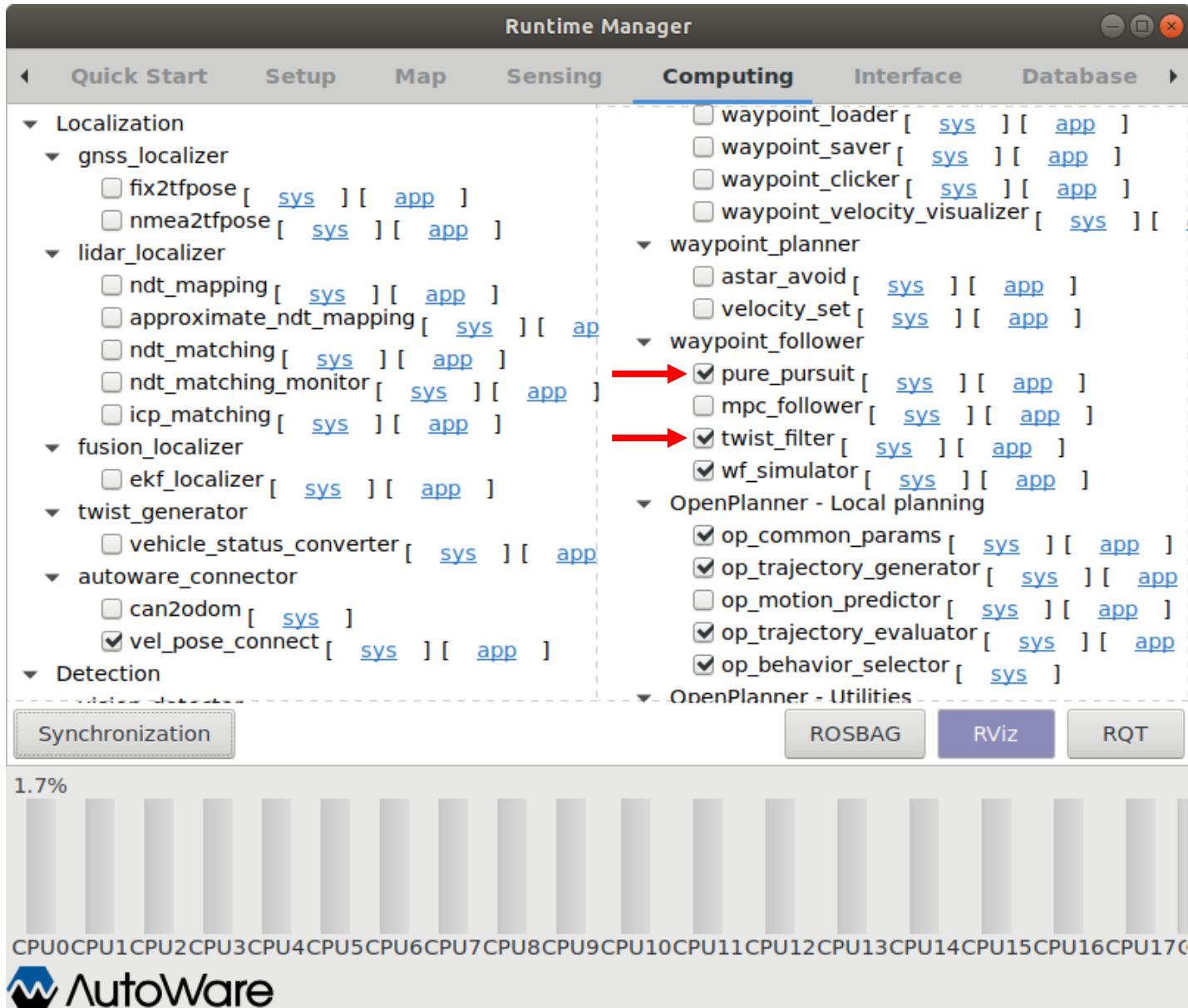
Step 4-7: Set start pose once again



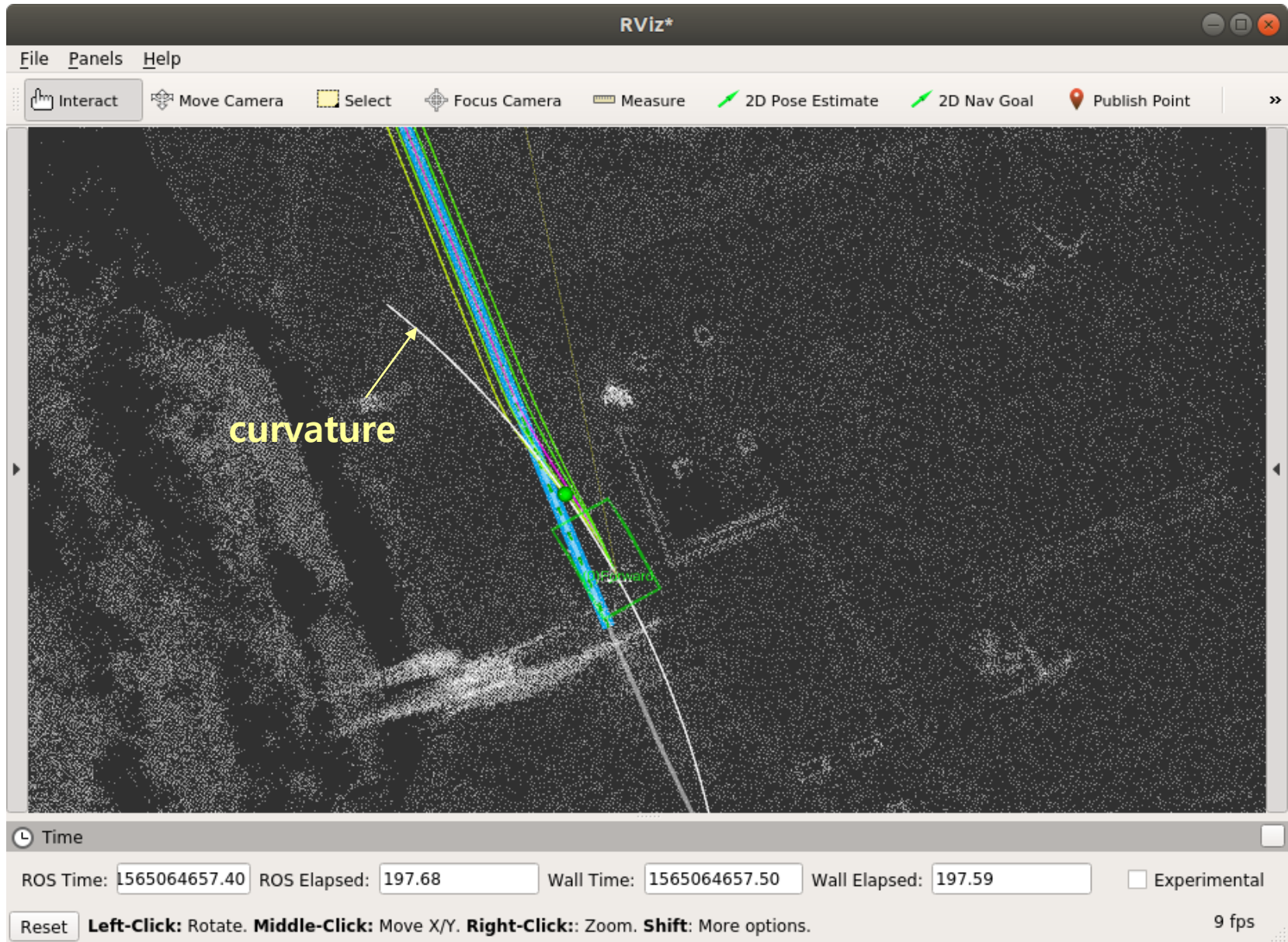
Step 4-7: Set start pose once again



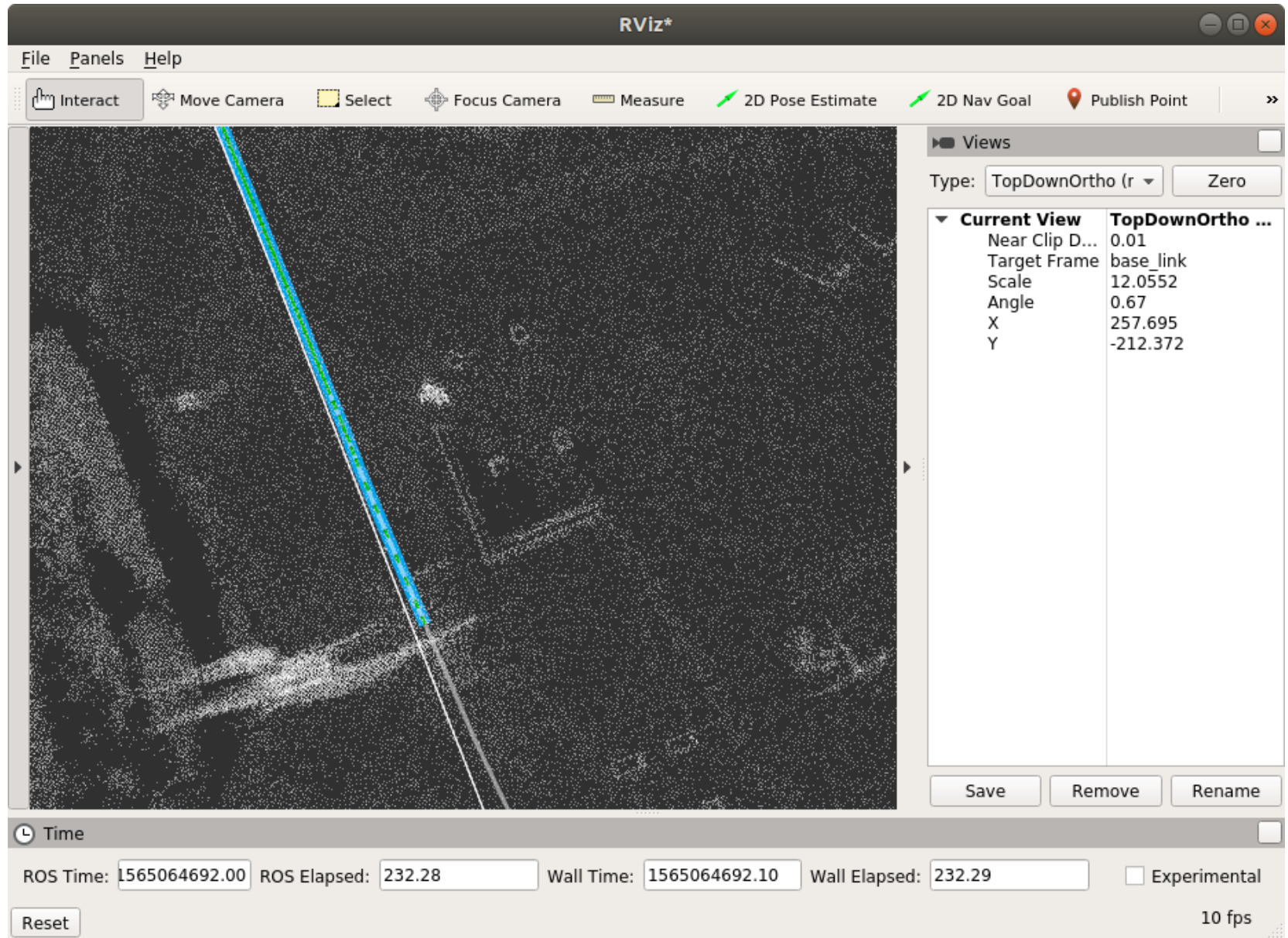
Step 4-8: Launch pure_pursuit & twist_filter



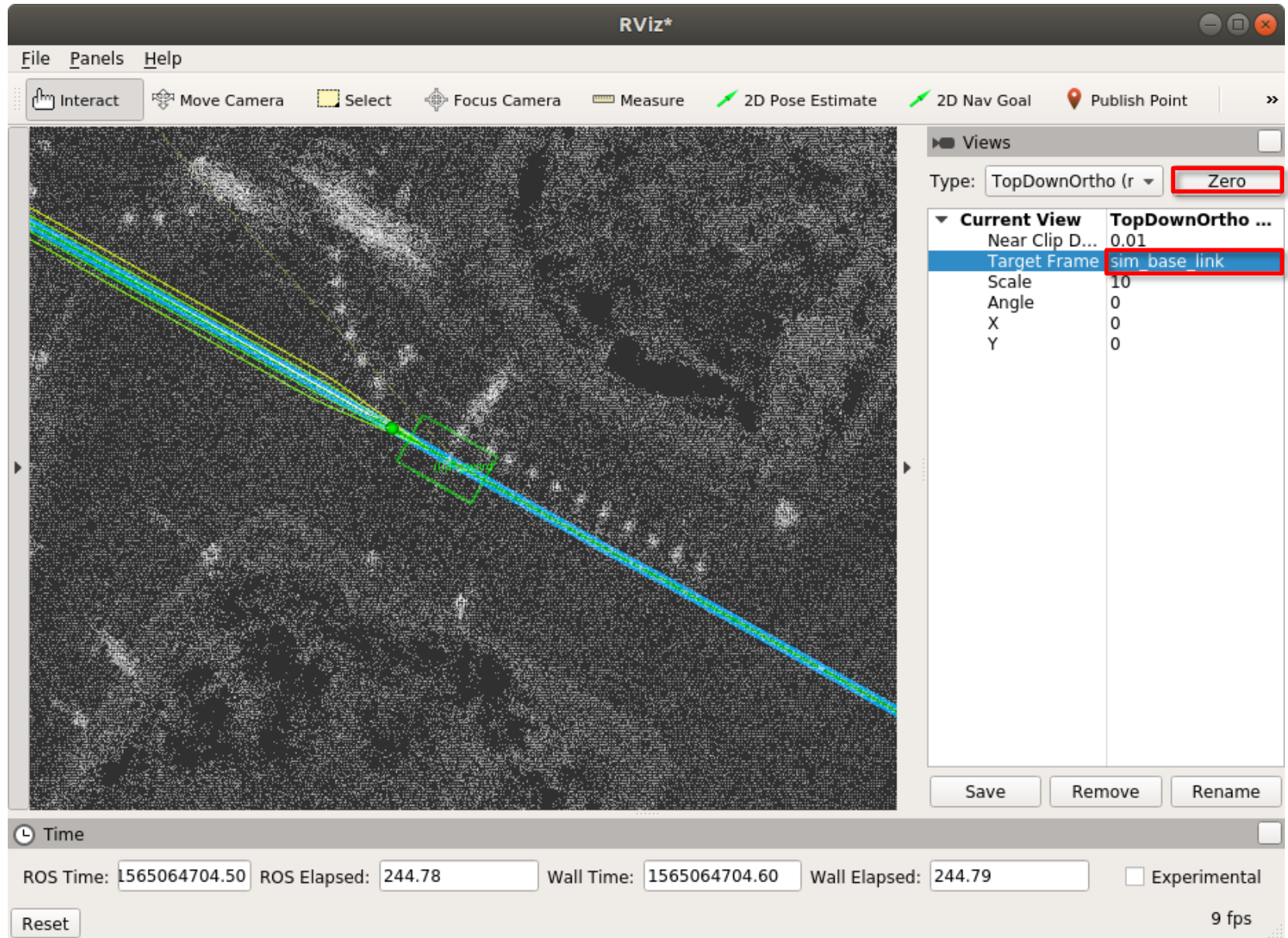
Step 4-8: Launch pure_pursuit & twist_filter



Step 4-8: Vehicle is gone!



Step 4-9: Change Target Frame



Step 4-10: Quit

- ❖ kill rviz by clicking x button
- ❖ kill runtime_manager by clicking x button
- ❖ enter ^C on the controlling terminal

```

started roslaunch server http://rubicom-MS-7B09:40243/

SUMMARY
=====

PARAMETERS
* /roscdistro: melodic
* /rosversion: 1.14.3

NODES
/
  run (runtime_manager/run)

auto-starting new master
process[master]: started with pid [1745]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 7c39d67c-b803-11e9-96f9-00e06333e8fb
process[rosout-1]: started with pid [1756]
started core service [/rosout]
process[run-2]: started with pid [1762]
[run-2] process has finished cleanly
log file: /home/autoware/.ros/log/7c39d67c-b803-11e9-96f9-00e06333e8fb/run-2*.log
^C[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
autoware@rubicom-MS-7B09:~/Autoware$

```

rosvag Play

Step 1: Run 'local-kinetic' container

“Simulation in Autoware”의 step 1을 먼저 수행함

- ❖ [host] \$ cd ~/docker/generic
- ❖ [host] \$ sudo ./run.sh -s
 - # 'local-kinetic' or 'local-kinetic-cuda' 이미지 기반 container 실행
- ❖ 또는 [host] \$ sudo ./run_aw.sh local-kinetic
 - # run_aw.sh 파일 안에 '—runtime=nvidia' 라인을 comment out 처리

또는 제공된 USB Disk 안에 prebuilt docker image를 복사해서 사용함

- ❖ [host] \$ cp /media/autoware/AUTOWARE/lab-kinetic.tar.gz ~/docker/generic
- ❖ [host] \$ cd ~/docker/generic
- ❖ [host] \$ sudo docker load < lab-kinetic.tar.gz
- ❖ [host] \$ sudo docker images
 - 'autoware/autoware:lab-kinetic' 라는 이미지가 생성됨.
 - 이 이미지는 다음 step 2와 step 3 실행 결과가 이미 반영되었음.
 - 따라서 이 이미지를 사용하면 step 4로 점프할 것.
- ❖ [host] \$ sudo ./run_aw.sh lab-kinetic

Step 2: Rebuild and install Autoware packages

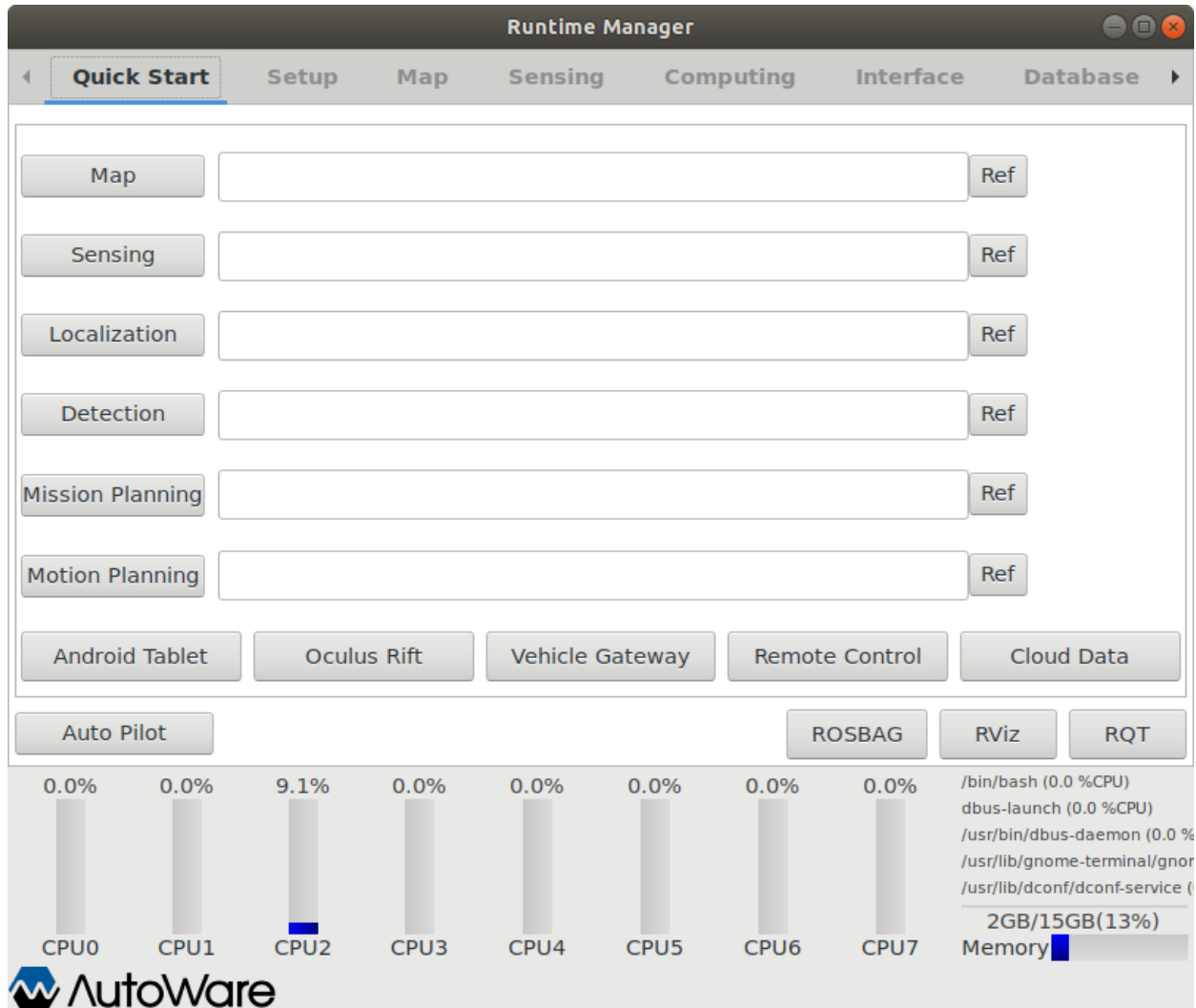
- ❖ \$ cd # change to home directory
- ❖ \$ cp shared_dir/src/autoware/utilities/runtime_manager/scripts/run Autoware/src/autoware/utilities/runtime_manager/scripts/run
- ❖ \$ cd Autoware/build/**runtime_manager**
- ❖ \$ make install # install 'run' file → "Simulation in Autoware"의 step 3와 동일
- ❖ \$ cd # change to home directory
- ❖ \$ cp shared_dir/src/autoware/core_perception/lidar_localizer/launch/ndt_matching.launch Autoware/src/autoware/core_perception/lidar_localizer/launch/ndt_matching.launch
- ❖ \$ cd Autoware/build/**lidar_localizer**
- ❖ \$ make install # rebuild and install ndt_matching
- ❖ \$ cd # change to home directory
- ❖ \$ cp shared_dir/src/autoware/core_perception/lidar_euclidean_cluster_detect/launch/lidar_euclidean_cluster_detect.launch Autoware/src/autoware/core_perception/lidar_euclidean_cluster_detect/launch/lidar_euclidean_cluster_detect.launch
- ❖ \$ cd Autoware/build/**lidar_euclidean_cluster_detect**
- ❖ \$ make install # install 'lidar_euclidean_cluster_detect.launch' file

Step 3: Prepare EgoCar.csv

- ❖ `$ cd`
- ❖ `$ mkdir -p autoware_openplanner_logs/SimulationData`
- ❖ `$ cp shared_dir/autoware_openplanner_logs/SimulationData/EgoCar.csv autoware_openplanner_logs/SimulationData/`

Step 4: Launch runtime_manager

- ❖ \$ cd Autoware
- ❖ \$ roslaunch runtime_manager runtime_manager.launch



Step 4-1: Load, Play, and Pause rosbag

Runtime Manager

Setup Map Sensing Computing Interface Database **Simulation**

(1) → `/home/autoware/shared_dir/rosbag_demo/kcity_20190807_lidar+gnss.bag` Ref

Rate: Start Time (s): **(2)** `30` ☐ Repeat

(3) → **Play** **Stop** **Pause** **(4)** → Playing... 0% 0/366

path: `/home/autoware/shared_dir/rosbag_demo/kcity_20190807_lidar+gnss.bag`
 version: 2.0
 duration: 6:36s (396s)
 start: Aug 07 2019 05:29:19.59 (1565155759.59)
 end: Aug 07 2019 05:35:55.96 (1565156155.96)
 size: 15.5 GB
 messages: 325025
 compression: none [7928/7928 chunks]
 types: geometry_msgs/PoseStamped [d3812c3cbc69362b77dc0b19b345f8f5]
 sensor_msgs/NavSatFix [2d3a8cd499b9b4a0249fb98fd05cfa48]
 sensor_msgs/PointCloud2 [1158d486dd51d683ce2f1be655c3c181]
 topics: /fix 158549 msgs : sensor_msgs/NavSatFix
 /gnss_pose 158549 msgs : geometry_msgs/PoseStamped
 /points_raw 7927 msgs : sensor_msgs/PointCloud2

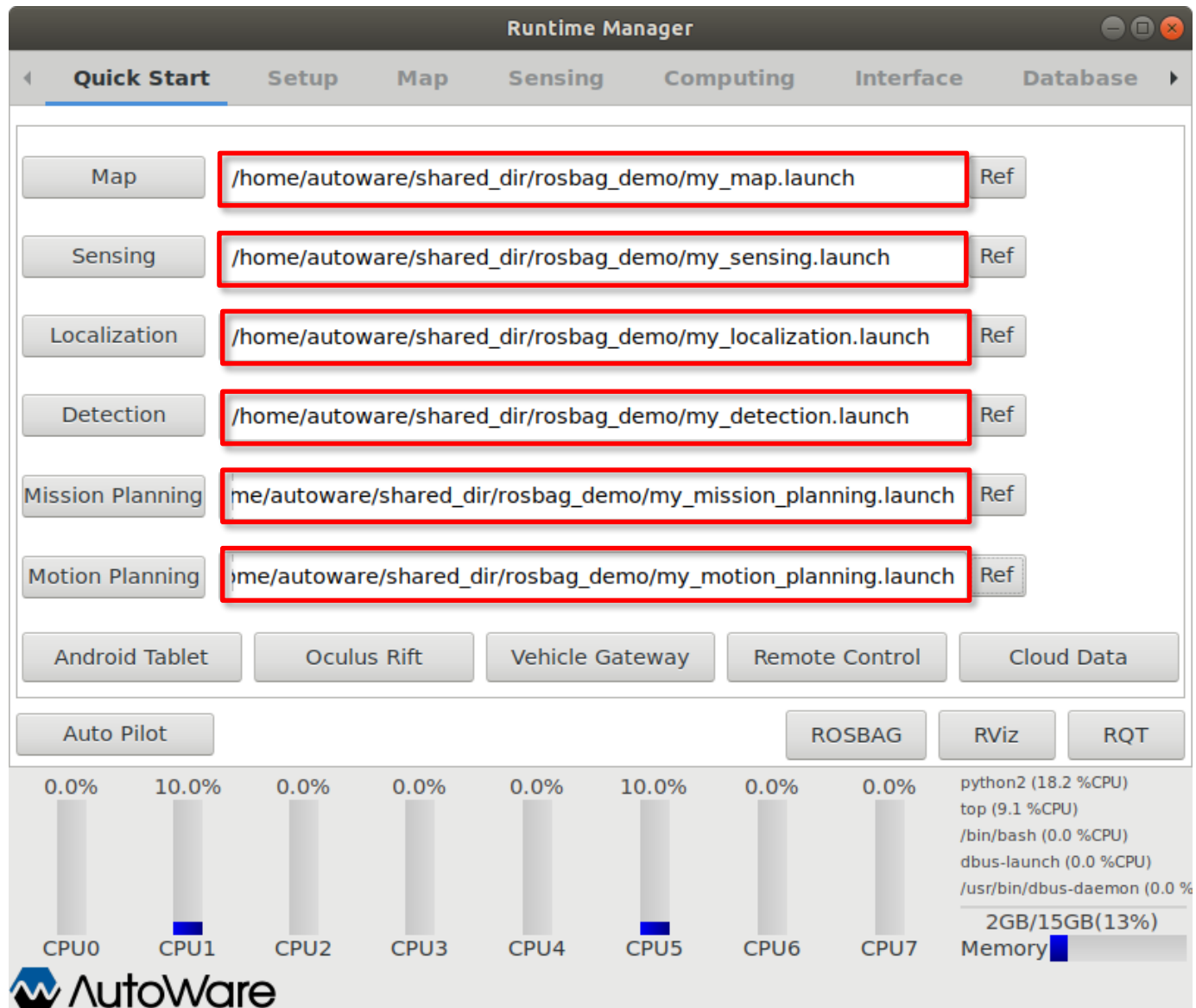
Gazebo LGSVL Simulator ROSBAG RViz RQT

CPU0: 0.0% CPU1: 0.0% CPU2: 0.0% CPU3: 11.1% CPU4: 9.1% CPU5: 10.0% CPU6: 0.0% CPU7: 0.0%

python2 (18.2 %CPU)
 /bin/bash (0.0 %CPU)
 dbus-launch (0.0 %CPU)
 /usr/bin/dbus-daemon (0.0 %CPU)
 /usr/lib/gnome-terminal/gnome-terminal (0.0 %CPU)
 2GB/15GB(13%)
 Memory

AutoWare

Step 4-2: Select Launch Files for Quick Start



Step 4-3: Launch Map and Localization

Runtime Manager

Quick Start Setup Map Sensing Computing Interface Database

(1) → Map /home/autoware/shared_dir/rosbag_demo/my_map.launch Ref

(2) → Localization /home/autoware/shared_dir/rosbag_demo/my_localization.launch Ref

Sensing /home/autoware/shared_dir/rosbag_demo/my_sensing.launch Ref

Detection /home/autoware/shared_dir/rosbag_demo/my_detection.launch Ref

Mission Planning /home/autoware/shared_dir/rosbag_demo/my_mission_planning.launch Ref

Motion Planning /home/autoware/shared_dir/rosbag_demo/my_motion_planning.launch Ref

Android Tablet Oculus Rift Vehicle Gateway Remote Control Cloud Data

Auto Pilot ROSBAG RViz RQT

9.1% 9.1% 0.0% 0.0% 0.0% 0.0% 0.0% 9.1%

CPU0 CPU1 CPU2 CPU3 CPU4 CPU5 CPU6 CPU7

python2 (18.2 %CPU)
/opt/ros/melodic/lib/rosbag/p
/home/autoware/Autoware/ir
/opt/ros/melodic/lib/robot_st
/home/autoware/Autoware/ir
2GB/15GB(17%)
Memory

AutoWare

Step 4-4: Unpause rosbag and Launch rviz

Runtime Manager

Setup Map Sensing Computing Interface Database **Simulation**

/home/autoware/shared_dir/rosbag_demo/kcity_20190807_lidar+gnss.bag Ref

Rate: Start Time (s): 30 Repeat

Play Stop **Pause** (1) Playing... 5% 16/366

path: /home/autoware/shared_dir/rosbag_demo/kcity_20190807_lidar+gnss.bag
 version: 2.0
 duration: 6:36s (396s)
 start: Aug 07 2019 05:29:19.59 (1565155759.59)
 end: Aug 07 2019 05:35:55.96 (1565156155.96)
 size: 15.5 GB
 messages: 325025
 compression: none [7928/7928 chunks]
 types: geometry_msgs/PoseStamped [d3812c3cbc69362b77dc0b19b345f8f5]
 sensor_msgs/NavSatFix [2d3a8cd499b9b4a0249fb98fd05cfa48]
 sensor_msgs/PointCloud2 [1158d486dd51d683ce2f1be655c3c181]
 topics: /fix 158549 msgs : sensor_msgs/NavSatFix
 /gnss_pose 158549 msgs : geometry_msgs/PoseStamped
 /points_raw 7927 msgs : sensor_msgs/PointCloud2

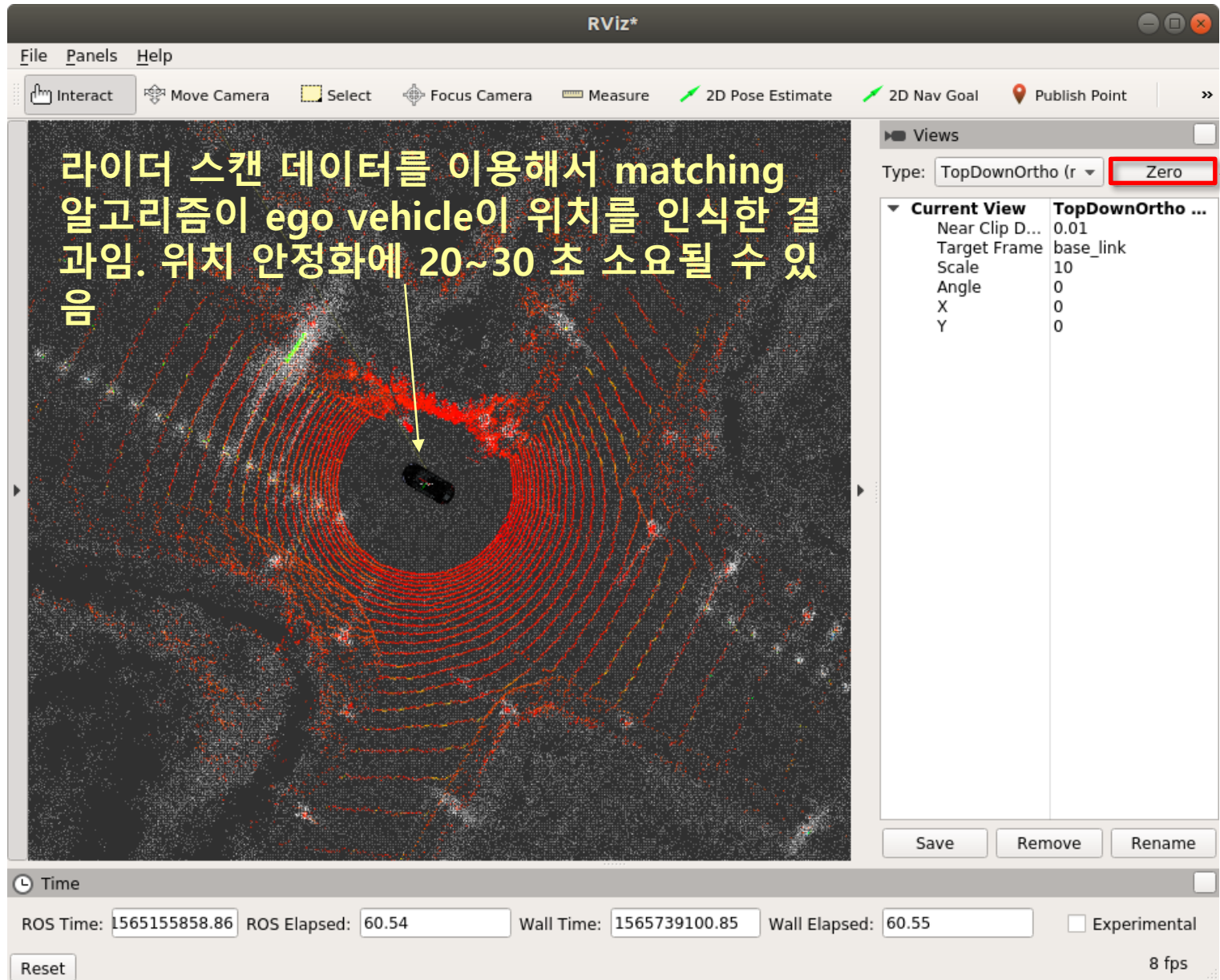
Gazebo LGSVL Simulator ROSBAG **RViz** (2) RQT

CPU0: 11.1% CPU1: 25.0% CPU2: 20.0% CPU3: 81.8% CPU4: 20.0% CPU5: 30.0% CPU6: 8.3% CPU7: 0.0%

/home/autoware/Autoware/ir
 python2 (90.0 %CPU)
 /home/autoware/Autoware/ir
 /opt/ros/melodic/lib/rosbag/p
 /opt/ros/melodic/lib/robot_st
 4GB/15GB(31%)
 Memory

AutoWare

Step 4-4: Unpause rosbag and Launch rviz



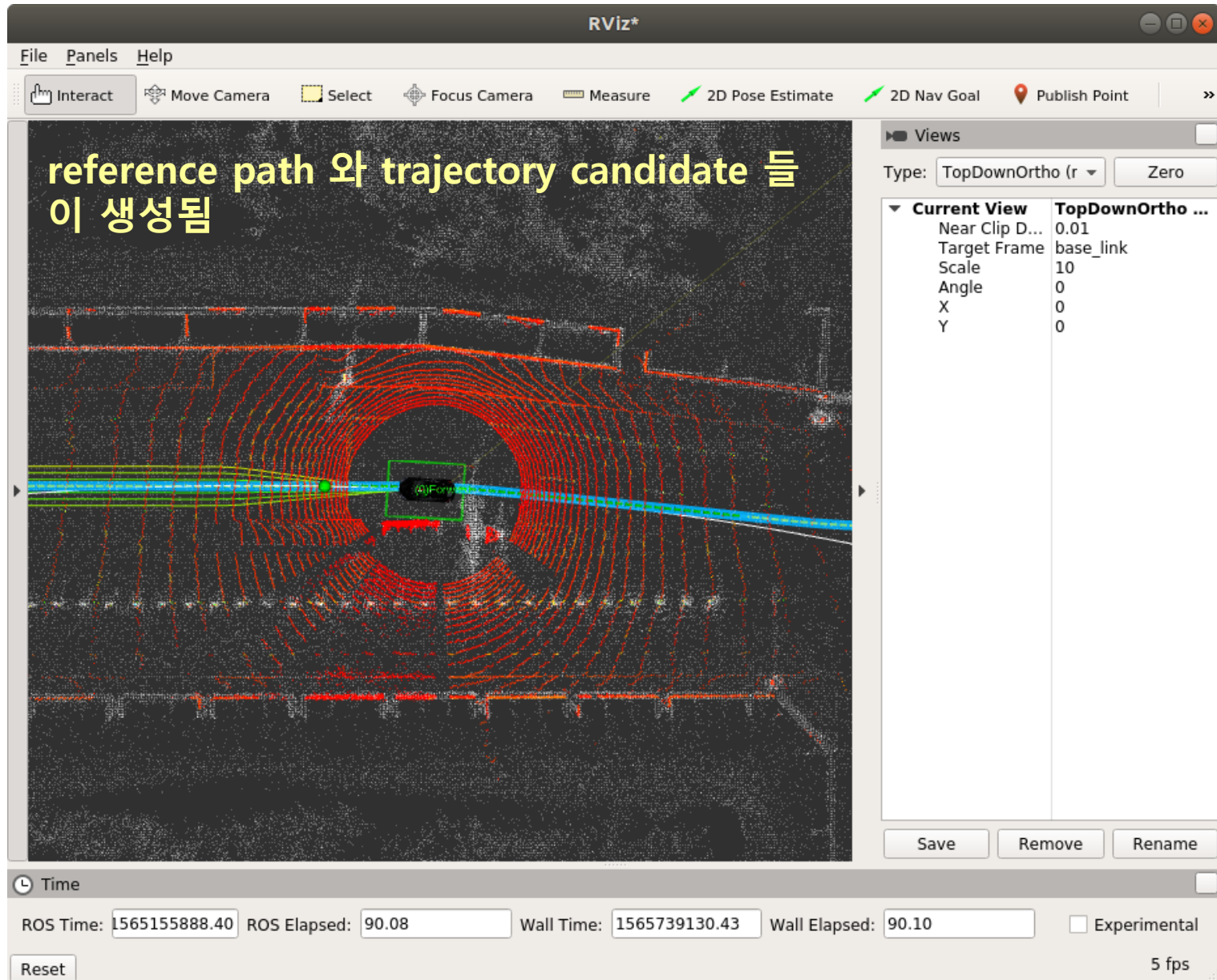
Step 4-5: Launch Global and Local Planner

(1) →

(2) →



Step 4-5: Launch Global and Local Planner



Step 4-6: Launch Sensing and Detection

Runtime Manager

Quick Start Setup Map Sensing Computing Interface Database

Map /home/autoware/shared_dir/rosbag_demo/my_map.launch Ref OK

(1) → Sensing /home/autoware/shared_dir/rosbag_demo/my_sensing.launch Ref

Localization /home/autoware/shared_dir/rosbag_demo/my_localization.launch Ref 47 ms

(2) → Detection /home/autoware/shared_dir/rosbag_demo/my_detection.launch Ref

Mission Planning /home/autoware/shared_dir/rosbag_demo/my_mission_planning.lau Ref

Motion Planning /home/autoware/shared_dir/rosbag_demo/my_motion_planning.laur Ref

Android Tablet Oculus Rift Vehicle Gateway Remote Control Cloud Data

Auto Pilot ROSBAG RViz RQT

CPU0 72.7% CPU1 50.0% CPU2 70.0% CPU3 80.0% CPU4 45.5% CPU5 0.0% CPU6 66.7% CPU7 54.5%

7GB/15GB(47%) Memory

AutoWare

Step 4-6: Launch Sensing and Detection

