

Sensors Recording System Using Ros1 and Ros2

Documentation File

Avi Tzahi

Shaked Nathan

GitHub Link:

https://github.com/shakednathan/RQt_synced_recording_system

Contents

Contents	2
Ros1 noetic installation:	3
Ros2 foxy installation:	4
ROS1 Bridge Installation	6
Delphi ESR:.....	7
IR Camera:	9
Inertiallabs INS:.....	11
Velodyne LIDAR	13
Innoviz LIDAR.....	15
RGB Camera.....	17
Rosbag to csv	19
Bag to Images script	20
Recording System Plugin	21
Sensors Synchronizer Node.....	24
Unified Launch File + System Bringup	25
Adding sensors to the ROS1 launch file.....	25
Using the Recording System	26
Recording.....	26
Viewing a Recording	27

Ros1 noetic installation:

Setup your computer to accept software from packages.ros.org.:

```
$ 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 install curl # if you haven't already installed curl

$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
apt-key add -
```

Installation:

```
$ sudo apt update

$ sudo apt install ros-noetic-desktop-full
```

Environment setup: You must source this script in every **bash** terminal you use ROS in.

```
$ source /opt/ros/noetic/setup.bash
```

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, `roscpp` is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
$ sudo apt install python3-roscpp python3-roscpp-generator
python3-wstool build-essential
```

Before you can use many ROS tools, you will need to initialize ROSdep. ROSdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed `roscpp`, do so as follows.

```
$ sudo apt install python3-roscpp
```

With the following, you can initialize `roscpp`.

```
$ sudo roscpp init

$ roscpp update
```

Ros1 noetic is installed on your machine.

Ros2 foxy installation:

Make sure you have a locale which supports UTF-8. If you are in a minimal environment (such as a docker container), the locale may be something minimal like POSIX. We test with the following settings. However, it should be fine if you're using a different UTF-8 supported locale.

```
$ locale # check for UTF-8

$ sudo apt update && sudo apt install locales

$ sudo locale-gen en_US en_US.UTF-8

$ sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8

$ export LANG=en_US.UTF-8

$ locale # verify settings
```

You will need to add the ROS 2 apt repository to your system.

First ensure that the Ubuntu Universe repository is enabled.

```
$ sudo apt install software-properties-common

$ sudo add-apt-repository universe
```

Now add the ROS 2 GPG key with apt.

```
$ sudo apt update && sudo apt install curl

$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
```

Then add the repository to your sources list.

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-
archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release &&
echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list >
/dev/null
```

Update your apt repository caches after setting up the repositories.

```
$ sudo apt update
```

ROS 2 packages are built on frequently updated Ubuntu systems. It is always recommended that you ensure your system is up to date before installing new packages.

```
$ sudo apt upgrade
```

Desktop Install (Recommended): ROS, RViz, demos, tutorials.

```
$ sudo apt install ros-foxy-desktop python3-argcomplete
```

Development tools: Compilers and other tools to build ROS packages

```
$ sudo apt install ros-dev-tools
```

Set up your environment by sourcing the following file.

```
$ # Replace ".bash" with your shell if you're not using bash
```

```
$ # Possible values are: setup.bash, setup.sh, setup.zsh
```

```
$ source /opt/ros/foxy/setup.bash
```

Ros2 foxy is installed on your machine.

If you need to uninstall ROS 2 or switch to a source-based install once you have already installed from binaries, run the following command:

```
$ sudo apt remove ~/ros-foxy-* && sudo apt autoremove
```

You may also want to remove the repository:

```
$ sudo rm /etc/apt/sources.list.d/ros2.list
```

```
$ sudo apt update
```

```
$ sudo apt autoremove
```

```
$ # Consider upgrading for packages previously shadowed.
```

```
$ sudo apt upgrade
```

ROS1 Bridge Installation

If you added any ROS sourcing lines to your bashrc file, comment them out first. Then, in a new terminal (so the bashrc changes will be applied) install ROS1 Bridge:

```
$ sudo apt install ros-foxy-ros1-bridge
```

To use the bridge:

1. Run roscore in a terminal with ROS1 sourced (roscore is run automatically upon launching a launch file, but can be also run manually)
2. In another terminal, source ROS1 and then ROS2
3. Run the following command in the terminal with both distros sourced:
`$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics`
4. Now ROS1 nodes should be seen in ROS2 and vice versa. To check that:
 - a. Open two terminals and source ROS1 in them. In one of them, run:

```
$ roscore
```

- b. Open another terminal and source both ROS distros (like in step 2), then run the bridge command from step 3
 - c. Open a fourth terminal, and source ROS2. Then run in the ROS1 terminal where roscore is not running:

```
$ rosrunc rospack_tutorials talker
```

- d. In the terminal with ROS2 source, run "ros2 topic list". You should see the "chatter" topic. Also, you can run:

```
$ ros2 run demo_nodes_cpp listener
```

And see if you get any reaction. You should get what's sent on the "chatter" topic.

Delphi ESR:

From the home directory:

```
$ cd ros_catkin_ws/src/
```

Then, git clone to the relevant package of radar interface:

```
$ git clone https://bitbucket.org/unizg-fer-lamor/radar\_interface.git
```

Then, build the package from your workspace:

```
$ cd ../
```

```
$ catkin_make
```

```
$ source devel/setup.bash
```

To read the radar data we have to setup the can:

```
$ sudo modprobe can
```

```
$ sudo modprobe can_raw
```

```
$ sudo ip link set can0 type can bitrate 500000
```

```
$ sudo ip link set up can0
```

To check our connection with can0 we have to run this command and view if we get data:

```
$ candump can0
```

How to view the radar data:

```
$ roslaunch radar_interface delphi_esr_can.launch
```

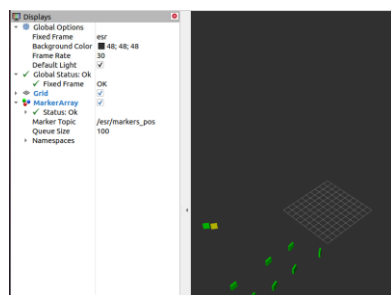
Then, to view the radar data in rviz of ros1 noetic:

```
$ rosrun rviz rviz
```

Then, add visualization of **MarkerArray** to the rviz window.

We are interested in the topic **/esr/markers_pos** to view the radar points around the car and fixed frame of **esr**.

The final result:



To view the radar data in rviz of ros2 foxy:

First, in one terminal window, source ROS1 and run a ROS master:

```
$ roscore
```

Then, in another terminal, source both ROS1 and ROS2 and run the node of ROS1 bridge:

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

Open two more terminals, and source ROS1 in one and ROS2 in the other. In the one with ROS1 sourced, run the radar node. In the one with ROS2 sourced, open RVIZ2:

```
$ ros2 run rviz2 rviz2
```

After that, add visualization of **MarkerArray** to the rviz window.

We are interested in the topic **/esr/markers_pos** to view the radar points around the car and fixed frame of **esr**.

IR Camera:

Change directory to your ros1 workspace (e.g. if your workspace is in ~/ros_catkin_ws cd to there):

```
$ cd ~/ros_catkin_ws/src/
```

Then, git clone to the relevant package of flir boson usb camera:

```
$ git clone https://github.com/astuff/flir\_boson\_usb.git
```

then, build the package from your workspace:

```
$ cd ../
```

```
$ catkin_make
```

```
$ source devel/setup.bash
```

how to view the IR camera data:

```
$ roslaunch flir_boson_usb flir_boson.launch
```

then, to view the IR camera data in rviz of ros1 noetic:

```
$ rosrun rviz rviz
```

then, add visualization of **image** to the rviz window.

We are interested in the topic **/flir_boson/image_raw** to view the image.

The final result:



To view the IR camera data in rviz of ros2 foxy:

First, in one terminal window, source ROS1 and run a ROS master:

```
$ roscore
```

Then, in another terminal, source both ROS1 and ROS2 and run the node of ros1 bridge:

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

In a third terminal, source only ROS2 and open rviz2:

```
$ ros2 run rviz2 rviz2
```

In a fourth terminal source ROS1 and run the FLIR camera node.

Add a visualization of **image** to the rviz window.

We are interested in the topic **/flir_boson/image_raw** to view the image.

Inertiallabs INS:

Change directory to your ros1 workspace (e.g. if your workspace is in ~/ros_catkin_ws cd to there):

```
$ cd ~/ros_catkin_ws/src/
```

then git clone to the relevant package of Inertiallabs INS:

```
$ git clone https://us.inertiallabs.com:31443/scm/ins/inertiallabs-ros-pkgs.git
```

edit the following line in ~/ros_catkin_ws/src/inertiallabs-ros-pkgs/inertiallabs_ins/src/il_ins.cpp :

from:

```
np.param<std::string>("ins_url", port, "serial:/dev/ttyUSB0:460800");
```

to:

```
np.param<std::string>("ins_url", port, "serial:/dev/ttyS3:115200");
```

then, build the package from your workspace:

```
$ cd ../
```

```
$ catkin_make
```

```
$ source devel/setup.bash
```

the device is connected to COM4 (physical and also could be seen in windows10).

In our linux (ubuntu) system it is ttyS3.

so we should expect to read the searial data via:

```
$ /dev/ttyS3
```

Set-up user permissions:

add user to dialout group: (not sure if necessary)

find out your user:

```
$ whoami
```

```
$ sudo adduser administrator dialout
```

*("administrator" is our username)

make sure it worked

```
$ groups administrator
```

Set-up user permission for serial connection:

Might need to do this every time system boots

```
$ sudo chmod 666 /dev/ttyS3
```

Now, installation is complete.

How to View the INS Data:

```
$ rosrn inertiallabs_ins il_ins
```

then use rostopic echo on the relevant topics – for our purpose is **/Inertial_Labs/ins_data**.

Final result:

```
---
^Header:
  seq: 297
  stamp:
    secs: 1668889136
    nsecs: 525580346
  frame_id: "F2001211"
GPS_INS_Time: 0.0
GPS_IMU_Time: 0.0
GPS_mSOW:
  data: 591554500
LLH:
  x: 32.1100201
  y: 34.8068245
  z: 56.42
```

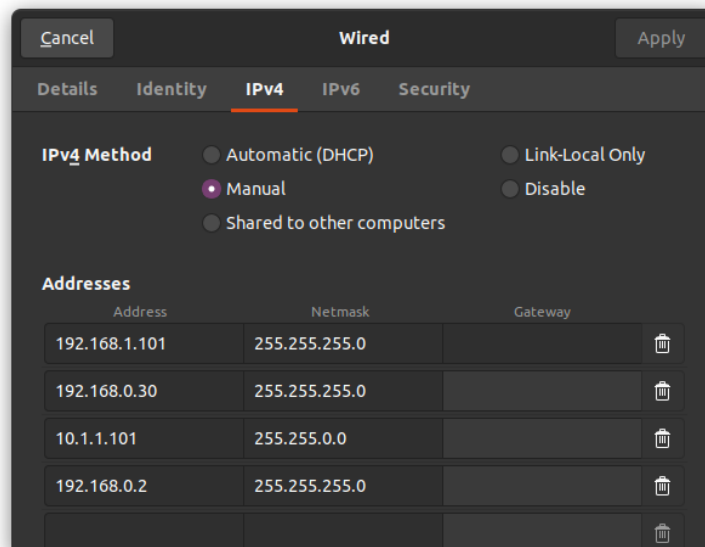
You can plug the (x, y) points in google maps to make sure the GPS coordinates are OK.

Velodyne LIDAR

This is a ROS2 package, so take note that things are a bit different than ROS1.

Network setup

The IP of our LIDAR is 192.168.1.201, which is the default IP, so to make the LIDAR communicate with the computer the following routing table should be set to one of the Ethernet switches:



Installation process

Open a terminal and type the following command:

```
$ sudo apt-get install ros-foxy-velodyne
```

You will be asked for your password, and maybe for confirmation of installation.

After that, download the package ZIP from the following link, or clone this repository:

<https://github.com/ros-drivers/velodyne/tree/ros2>

Make sure you get the ROS2 version. Cloning the repository “naively” actually downloaded a ROS1 version for us so take that into account if you want to use “git clone”.

Then extract the ZIP and move the “velodyne” folder to your ros2 workspace, into the src folder (for example, if your workspace is ~/dev_ws then move the folder to ~/dev_ws/src), if you downloaded the ZIP manually. Then cd in the terminal to your workspace (the folder that contains the src folder) and run the following command:

```
$ colcon build
```

This compiles the package. After that, run the following command:

```
$ source ./install/setup.bash
```

Then the velodyne package should be installed.

Viewing the LIDAR data

Open a terminal and run the following command:

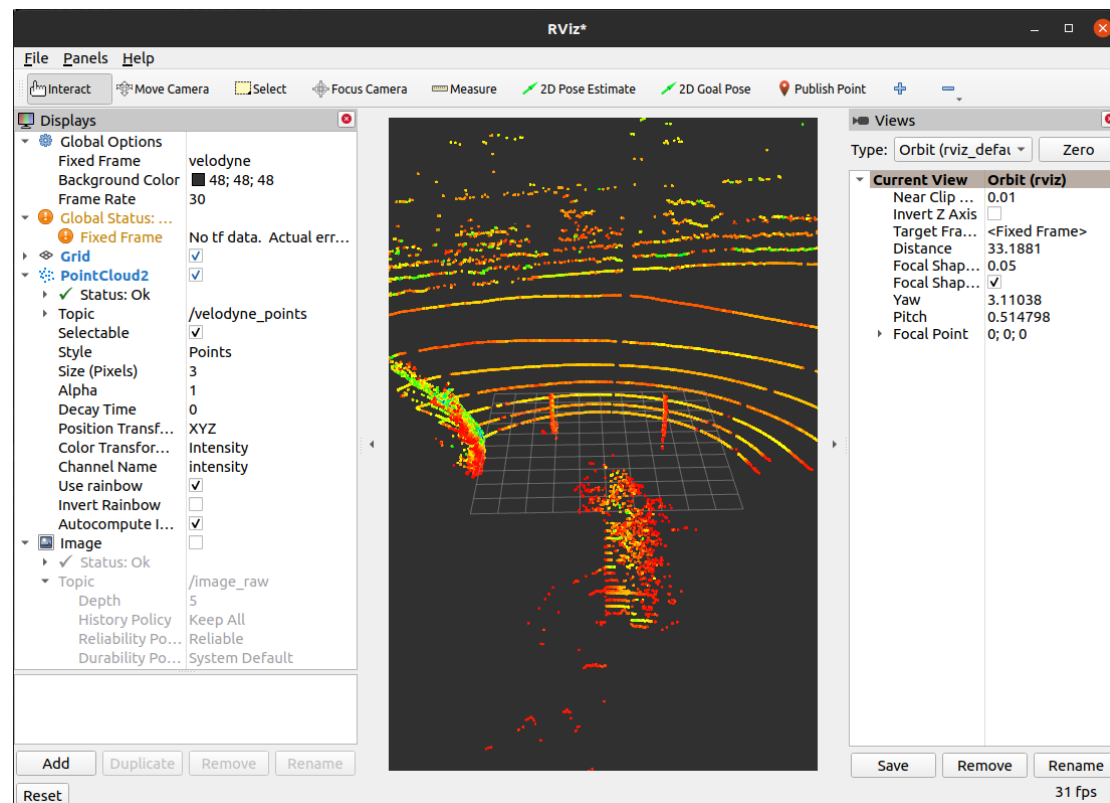
```
$ ros2 launch velodyne velodyne-all-nodes-VLP16-launch.py
```

To make sure you have the velodyne package installed you might want to use TAB key completion after you type “ros2 launch ve”. If nothing pops up (typically right after installation, in the terminal the installation ran from), close the terminal and open another one, and in the new one source the `./install/setup.bash` file from the root folder of your ros2 workspace.

You will typically see nothing - this is OK. The velodyne node is not a visualizer. To see the pointcloud of the LIDAR, run:

```
$ ros2 run rviz2 rviz2 -f velodyne
```

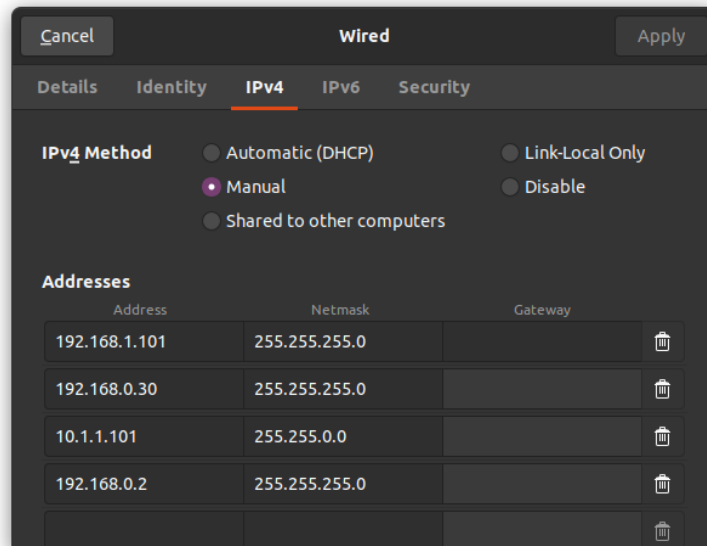
This will open a visualizer called RVIZ. In the sidebar of RVIZ, press “add”, then choose “pointcloud2”. In the “topic” field of the new pointcloud2 view, write “/velodyne_points”. This should yield a result like this:



The view can be rotated around (like standing in a point and turning to look around) by left-clicking and dragging, and zoomed in and out using the mouse wheel. Middle-clicking and dragging moves the image (like walking to a different standpoint).

Innoviz LIDAR

Set the routing table to accommodate the IP address of the Innoviz LIDAR. In our case, the Innoviz LIDAR has the address 192.168.0.3 so we set the routing table of the router it is connected to like so:



Change directory to your ros1 workspace (e.g. if your workspace is in ~/ros_catkin_ws cd to there):

```
$ cd ~/ros_catkin_ws/src/
```

Then, git clone to the relevant package of innoviz lidar:

```
$ git clone https://github.com/InnovizTechnologies/InnovizAPI.git
```

then, build the package from your workspace:

```
$ cd ../
```

```
$ echo /opt/ros/noetic/lib | sudo tee /etc/ld.so.conf.d/ros.conf
```

```
$ sudo ldconfig
```

```
$ catkin_make -DCMAKE_BUILD_TYPE=Release && sudo setcap  
cap_net_raw,cap_net_admin=eip ./devel/lib/innoviz_ros/Innoviz_one_Device
```

```
$ source devel/setup.bash
```

how to view the lidar data:

```
$ roslaunch innoviz_ros innoviz_ros.launch
```

The launch file already runs rviz configured to show the InnovizOne reflection0 point cloud. However, it is possible to run it separately, as well as show the other topics that the Innoviz_one_Device publish.

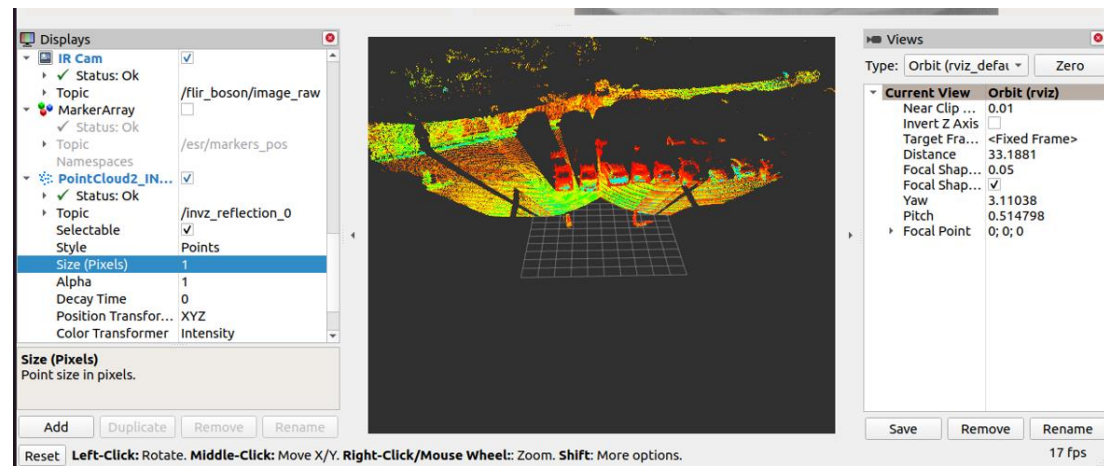
To do it separately, run rviz of ros1 noetic:

```
$ rosrun rviz rviz
```

then, add visualization of **pointcloud2** to the rviz window.

We are interested in the topic **/In vz_reflection_0** to view the points.

The final result:



To view the innoviz lidar in rviz of ros2 foxy:

First, in one terminal window, source ROS1 and run a ROS master:

```
$ roscore
```

Then, in another terminal, source both ROS1 and ROS2 and run the node of ros1 bridge:

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

In a third terminal, source only ROS2 and open rviz2:

```
$ ros2 run rviz2 rviz2
```

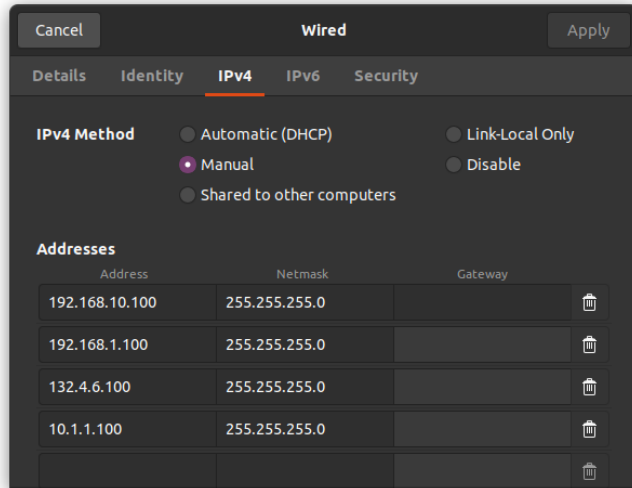
In a fourth terminal source ROS1 and run the innoviz lidar node.

Add a visualization of **pointcloud2** to the rviz window.

We are interested in the topic **/In vz_reflection_0** to view the points.

RGB Camera

First you need to set up a network switch for the camera. Our camera's IP is 192.168.10.150 so we set up the following routing table on one of our switches:



To install the drivers, download Vimba using the following link:

<https://www.alliedvision.com/en/products/vimba-sdk/#c1497>

Then, using the "Installing Vimba under Linux" file (found in the same link), follow the instructions to install Vimba.

Now download the code for the prosilica_driver package from here:

https://github.com/ros-drivers/prosilica_driver

Unzip it and move the folder to the ./src folder of your ROS1 workspace.

Do the same (download, unzip and move to ./src of the workspace) for the prosilica_gige_sdk, found here:

https://github.com/ros-drivers/prosilica_gige_sdk

Then open a terminal, source ROS1 and your workspace's setup.bash file, cd into there and run:

```
$ catkin_make
```

This should compile the newly added packages. Now close the terminal and source ROS1 and your workspace's setup.bash file.

Then enter the ./src folder of the workspace again, and in the following path:

```
./prosilica_driver/prosilica_camera/launch
```

edit the IP in the three launch files to the IP of the camera. For us it is 192.168.10.150.

You can now run the following:

```
$ roslaunch prosilica_camera generic.launch
```

You can verify that it works by typing

```
$ rostopic list
```

and making sure that there is an `/image_raw` node.

To view the image, you can use RVIZ in a similar way to how it is used with the IR camera.

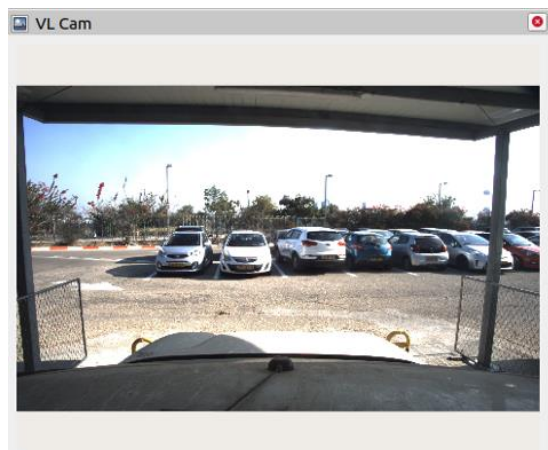
to view the radar data in rviz of ros1 noetic:

```
$ rosrn rviz rviz
```

then, add visualization of **image** to the rviz window.

We are interested in the topic **/image_raw** to view the image.

The final result:



To view the camera data in rviz of ros2 foxy:

First, in one terminal window, source ROS1 and run a ROS master:

```
$ roscore
```

Then, in another terminal, source both ROS1 and ROS2 and run the node of ros1 bridge:

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

In a third terminal, source only ROS2 and open rviz2:

```
$ ros2 run rviz2 rviz2
```

In a fourth terminal source ROS1 and run the camera node.

Add a visualization of **image** to the rviz window.

We are interested in the topic **/image_raw** to view the image.

Because Rviz does not support demosaic-ing color images, it shows the image as grayscale. This can be corrected by running, in another terminal with ROS2 sourced, the following node:

```
$ ros2 run image_proc image_proc
```

This will create a topic named `/image_color`, which Rviz can work with and display correctly.

Rosbag to csv

Change directory to your ros1 workspace (e.g. if your workspace is in ~/ros_catkin_ws cd to there):

```
$ cd ~/ros_catkin_ws/src/
```

Then, git clone to the relevant package of rosbag_to_csv:

```
$ git clone https://github.com/AtsushiSakai/rosbag\_to\_csv.git
```

then, build the package from your workspace:

```
$ cd ../
```

```
$ rosdep install -r --ignore-src --from-paths src
```

```
$ catkin_make
```

```
$ source devel/setup.bash
```

Start the node:

```
$ rosrun rosbag_to_csv rosbag_to_csv.py
```

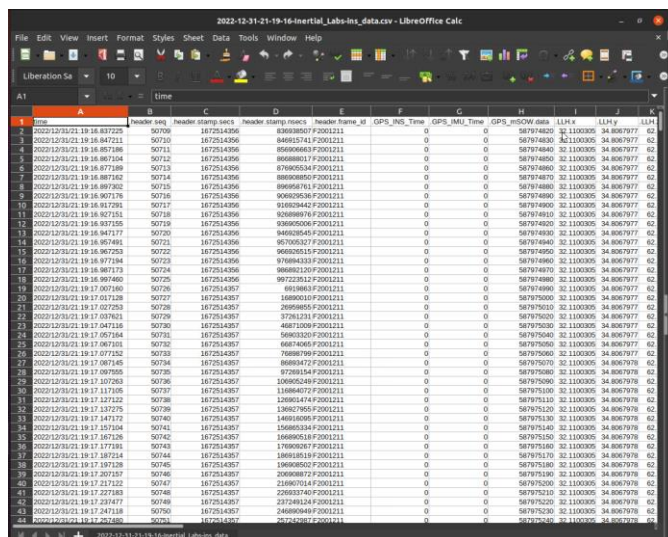
After run the node, you can select topics from the gui to save in csv files.

Wait some seconds and then a message "converting..." is displayed in the terminal.

When the finish convert message dialog is shown, CSV files are generated successfully in the current directory.

The CSV file name is same as (the bag file name)_(each selected topic name).csv.

The file can be opened using LibreOffice or any similar program:



	A	B	C	D	E	F	G	H	I	J	K
	time	header seq	header stamp	header topic	header frame_id	GPS_RNG_Time	GPS_IMG_Time	GPS_HRSW_data	L1H4	L1H5	L1H6
1	2022-12-31-19-16-837225	50709	1672543956	836898527.F2001211		0	0	567974830	32.1100305	34.8067977	62
2	2022-12-31-19-16-847211	50710	1672543956	84695174.F2001211		0	0	567974830	32.1100305	34.8067977	62
3	2022-12-31-19-16-857186	50711	1672543956	856906663.F2001211		0	0	567974840	32.1100305	34.8067977	62
4	2022-12-31-19-16-867164	50712	1672543956	86686017.F2001211		0	0	567974850	32.1100305	34.8067977	62
5	2022-12-31-19-16-877139	50713	1672543956	876815234.F2001211		0	0	567974860	32.1100305	34.8067977	62
6	2022-12-31-19-16-887102	50714	1672543956	886808852.F2001211		0	0	567974870	32.1100305	34.8067977	62
7	2022-12-31-19-16-897082	50715	1672543956	896846761.F2001211		0	0	567974880	32.1100305	34.8067977	62
8	2022-12-31-19-16-907176	50716	1672543956	906809536.F2001211		0	0	567974890	32.1100305	34.8067977	62
9	2022-12-31-19-16-917081	50717	1672543956	916829444.F2001211		0	0	567974890	32.1100305	34.8067977	62
10	2022-12-31-19-16-927151	50718	1672543956	926868876.F2001211		0	0	567974910	32.1100305	34.8067977	62
11	2022-12-31-19-16-937155	50719	1672543956	936855006.F2001211		0	0	567974920	32.1100305	34.8067977	62
12	2022-12-31-19-16-947177	50720	1672543956	946838544.F2001211		0	0	567974930	32.1100305	34.8067977	62
13	2022-12-31-19-16-957491	50721	1672543956	957005327.F2001211		0	0	567974940	32.1100305	34.8067977	62
14	2022-12-31-19-16-967253	50722	1672543956	966925153.F2001211		0	0	567974950	32.1100305	34.8067977	62
15	2022-12-31-19-16-977194	50723	1672543956	976884333.F2001211		0	0	567974960	32.1100305	34.8067977	62
16	2022-12-31-19-16-987173	50724	1672543956	986861220.F2001211		0	0	567974970	32.1100305	34.8067977	62
17	2022-12-31-19-16-997460	50725	1672543956	997238127.F2001211		0	0	567974980	32.1100305	34.8067977	62
18	2022-12-31-19-17-007180	50726	1672543957	00108663.F2001211		0	0	567974990	32.1100305	34.8067977	62
19	2022-12-31-19-17-017128	50727	1672543957	016800319.F2001211		0	0	567975000	32.1100305	34.8067977	62
20	2022-12-31-19-17-027253	50728	1672543957	026958955.F2001211		0	0	567975010	32.1100305	34.8067977	62
21	2022-12-31-19-17-037611	50729	1672543957	037223129.F2001211		0	0	567975020	32.1100305	34.8067977	62
22	2022-12-31-19-17-047136	50730	1672543957	046710089.F2001211		0	0	567975030	32.1100305	34.8067977	62
23	2022-12-31-19-17-057184	50731	1672543957	056853320.F2001211		0	0	567975040	32.1100305	34.8067977	62
24	2022-12-31-19-17-067101	50732	1672543957	066784661.F2001211		0	0	567975050	32.1100305	34.8067977	62
25	2022-12-31-19-17-077152	50733	1672543957	076888799.F2001211		0	0	567975060	32.1100305	34.8067977	62
26	2022-12-31-19-17-087145	50734	1672543957	086854737.F2001211		0	0	567975070	32.1100305	34.8067978	62
27	2022-12-31-19-17-097955	50735	1672543957	097881544.F2001211		0	0	567975080	32.1100305	34.8067978	62
28	2022-12-31-19-17-107263	50736	1672543957	108852649.F2001211		0	0	567975090	32.1100305	34.8067978	62
29	2022-12-31-19-17-117106	50737	1672543957	118864072.F2001211		0	0	567975100	32.1100305	34.8067978	62
30	2022-12-31-19-17-127122	50738	1672543957	128854744.F2001211		0	0	567975110	32.1100305	34.8067978	62
31	2022-12-31-19-17-137275	50739	1672543957	138857955.F2001211		0	0	567975120	32.1100305	34.8067978	62
32	2022-12-31-19-17-147172	50740	1672543957	148814008.F2001211		0	0	567975130	32.1100305	34.8067978	62
33	2022-12-31-19-17-157141	50741	1672543957	158835183.F2001211		0	0	567975140	32.1100305	34.8067978	62
34	2022-12-31-19-17-167126	50742	1672543957	168860518.F2001211		0	0	567975150	32.1100305	34.8067978	62
35	2022-12-31-19-17-177861	50743	1672543957	178880789.F2001211		0	0	567975160	32.1100305	34.8067978	62
36	2022-12-31-19-17-187114	50744	1672543957	188818513.F2001211		0	0	567975170	32.1100305	34.8067978	62
37	2022-12-31-19-17-197128	50745	1672543957	198808502.F2001211		0	0	567975180	32.1100305	34.8067978	62
38	2022-12-31-19-17-207187	50746	1672543957	208868871.F2001211		0	0	567975190	32.1100305	34.8067978	62
39	2022-12-31-19-17-217122	50747	1672543957	218807014.F2001211		0	0	567975200	32.1100305	34.8067978	62
40	2022-12-31-19-17-227166	50748	1672543957	228837512.F2001211		0	0	567975210	32.1100305	34.8067978	62
41	2022-12-31-19-17-237477	50749	1672543957	237489124.F2001211		0	0	567975220	32.1100305	34.8067978	62
42	2022-12-31-19-17-247118	50750	1672543957	246869449.F2001211		0	0	567975230	32.1100305	34.8067978	62
43	2022-12-31-19-17-257280	50751	1672543957	257242822.F2001211		0	0	567975240	32.1100305	34.8067978	62

Each row represents a message on the exported topic.

This method is suitable for exporting the INS and RADAR data from bag files.

Bag to Images script

Download the python script from the following link:

<https://gist.github.com/wngreene/835cda68ddd9c5416defce876a4d7dd9>

To make the script run in python3, add parentheses around the arguments of "print" and replace "python" in the first line (starts with "#!") to python3.

Also, add the following lines before the loop, so the script creates the output directory if it doesn't exist:

```
if not os.path.exists(args.output_dir):  
    os.mkdir(args.output_dir)
```

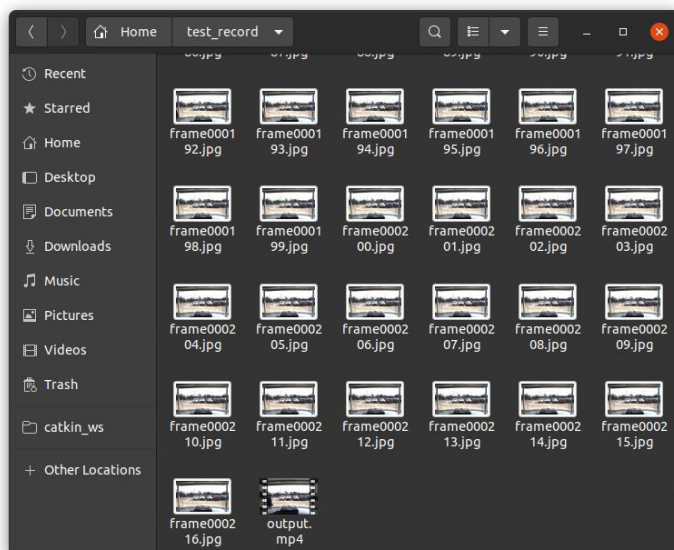
We commented the print line inside the loop to not flood the terminal with "Wrote image %d" lines. This is optional, but highly recommended.

To use the script, run the following command (from the directory of the script):

```
$ python3 bag_to_images.py <output folder path> <bag file path> <topic name>
```

Where <output folder path> is the relative path of the output directory (relative to the directory of the script), <bag file path> is the path to the bag file to convert, and <topic name> is the name of the topic to convert.

After running the script, a folder will be created. This is what should be in the folder:



The "output.mp4" file is not generated by the script – it is generated by FFmpeg, using the following command, run from the directory the images are in:

```
$ ffmpeg -framerate F -i frame%06d.jpg -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p output.mp4 #replace F with the frame rate of the sensor
```

Recording System Plugin

To install the plugin, download the following folder from our GitHub repository (link in the cover page):

- `rqt_car_sensors_rec`

Then move the folder into `<ros1_ws_path>/src` folder, and build the package using the following commands:

```
$ cd <ros1_ws_path>
```

```
$ catkin_make
```

Then, you can add the plugin to an RQt window and start working.

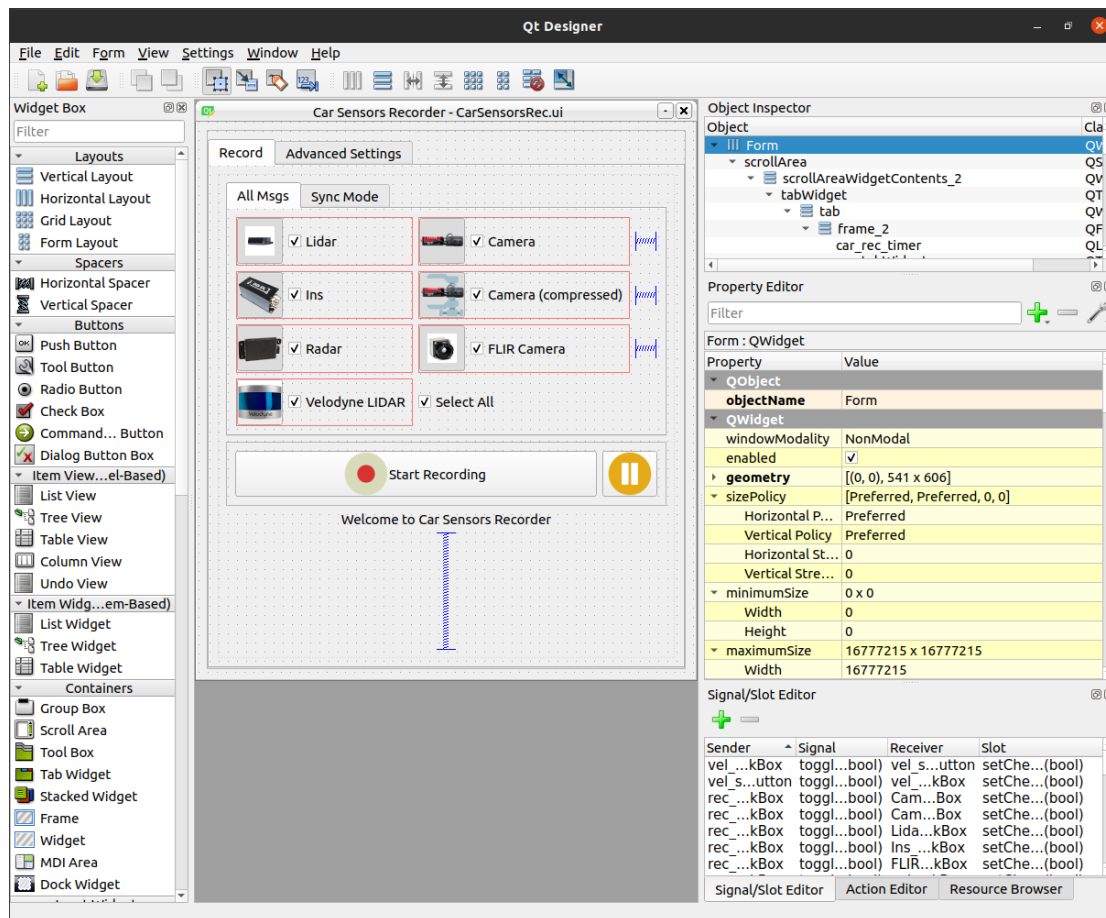
That is, assuming you use the exact same sensors we used. To add sensors to the recording list, the following changes to the python file (found inside the package, in the path `rqt_car_sensors_req/src/rqt_car_sensors_req/car_sensors_req.py`) are required:

1. Adding lines for sensor codes:
`NEW_SENSOR_CODE = a`
(Replace `a` with a number smaller than `SYNC_CODE`)
2. Adding lines for sensors sync code:
`NEW_SYNC_CODE = SYNC_CODE + NEW_SENSOR_CODE`
3. Adding entries for the new sensors in `code2sensor_dict`:
`NEW_SYNC_CODE: "sensortype"`
4. Adding entries for the relevant topics for each sensor in `code2topic_dict`:
`NEW_SENSOR_CODE: "/topic_name"`
5. Adding lines for each button and checkbox in the GUI:

```
self._widget.NewButton.clicked.connect\  
    (lambda: self.sensorClicked(NEW_SENSOR_CODE))  
self._widget.NewCheckBox.clicked.connect\  
    (lambda: self.sensorClicked(NEW_SENSOR_CODE))
```

Similarly, add similar lines for the sensor in sync mode (with `NEW_SYNC_CODE` instead of `NEW_SENSOR_CODE`), with the checkbox and button names that will be used in Qt Designer.

6. Adding the buttons and checkboxes in Qt Designer:
The Qt Designer window looks like this:



Widgets can be dragged from the left panel into the form in the middle. To edit a widget's parameters, click it and look for the parameter to change in the property editor on the right. In the bottom right, there is the signal/slot editor, where one can add signals that widgets can send to one another.

- Open resource/CarSensorsReq.ui with Qt Designer
- Press on the tab you want to edit (sync mode or normal mode)
- Move the "select all" checkbox out of the way to a new location.
- Add a "vertical layout" in the wanted place, and inside it, add a button and a checkbox. Change the names of the button and the checkbox to some meaningful names (remember you need to use these names from the code!) and set the button's size to 50x50 (using the panel on the right).
- Choose an image to use as the button's image and set it accordingly
- Set the checkbox's text to whatever you want to be displayed there, and remove the button's text.
- Add linkages between the parts: in the signal/slot editor of Qt Designer, add a line with the button as a sender, the checkbox as the receiver, set the signal to toggled(bool) and the slot to setChecked(bool). Add another line, reversing the roles of the checkbox and the button.
- Finally, for each checkbox, add a line in the signal/slot editor with the checkbox as a receiver and the "select all" checkbox (of the relevant tab) as the sender, with the same signal and slot.

The above instructions should be repeated for any added button, and the buttons must be also added to the python script.

7. Adding a line for each checkbox in the function `is_none_sensor_toggled`:
`new_sensor = self._widget.newCheckBox.isChecked()`
(a line should be added for sync mode as well, if using it)
Also, in the return statement in the end of the function, add the sensors to the Boolean expression.
8. Adding lines for each sensor and checkbox in `set_sensor_button_availability`:
`self._widget.NewButton.setEnabled(status)`
`self._widget.NewCheckBox.setEnabled(status)`

No need to do it for the sync mode (apparently)

Sensors Synchronizer Node

Download the following folders and files from our GitHub repository (link in the cover page):

- sensors_synchronizer

Move the folder to the src folder of your ros1 workspace and run catkin_make to build it (from a terminal directed to the root of your workspace)

Then, in the package, open the python script (in src/ of the package) named sensors_synchronizer.py, and add the following lines:

- In arg2topic_dict, add an entry for the sensor using the same name found under its code in the recording system python script
- Add the following condition in the "main" (top level code) of the script, replacing the "camera_comp" with your sensor's name and "image_raw/compressed" with the name of the relevant topic:

```
if "camera_comp" in topics_list:
    image_comp_pub = rospy.Publisher(
        'sensor_sync/image_raw/compressed', CompressedImage,
        queue_size=5)
    image_comp_sub = message_filters.Subscriber(
        'image_raw/compressed',
        CompressedImage)
```

```
msgs2indx["camera_comp"] = i
i += 1
sync_topics_list.append(image_comp_sub)
else:
```

```
    image_comp_pub = None
```

If there are multiple relevant topics, create a subscriber and publisher for each one, and append all of them to sync_topics_list.

- In the callback function, add a condition like the following one:

```
if vel_lidar_pub != None:
    vel_lidar_pub.publish(callback_msgs[msgs2indx["vel_lidar"]])
```

replacing "vel_lidar_pub" with the name of the publisher you created for that sensor, and "vel_lidar" with the name of the sensor. If there are multiple nodes, follow the example in the script of the INS.

The sync node is run automatically when starting a synchronized recording, so no need to run it manually.

Unified Launch File + System Bringup

Download the following files and folders from the project's page on GitHub (link in the cover page):

- rqt_car_sensors_rec
- rqt_ins_display
- rqt_bag_playback
- sensors_synchronizer
- ros1_nodes.launch
- bridge_setup.bash
- Car_Sensors_Recording.perspective

Before launching the system, open the launch file and replace the path to the RQt perspective file with the path of that file in your system. Any changes you make to the RQt window will be saved in this perspective file.

To launch the system, run the following command in a terminal with ROS1 and your workspace sourced:

```
$ roslaunch <path to file>/ros1_nodes.launch
```

The system will launch with everything in ROS1 (so, no ROS2 sensors yet).

To launch the ROS2 components, open two more terminals. In the first, run:

```
$ source <path to file>/bridge_setup.bash
```

This will open the bridge node.

In the second terminal, run your ROS2 sensors you want to use. For us, this is used to visualize the Velodyne LIDAR, so we run:

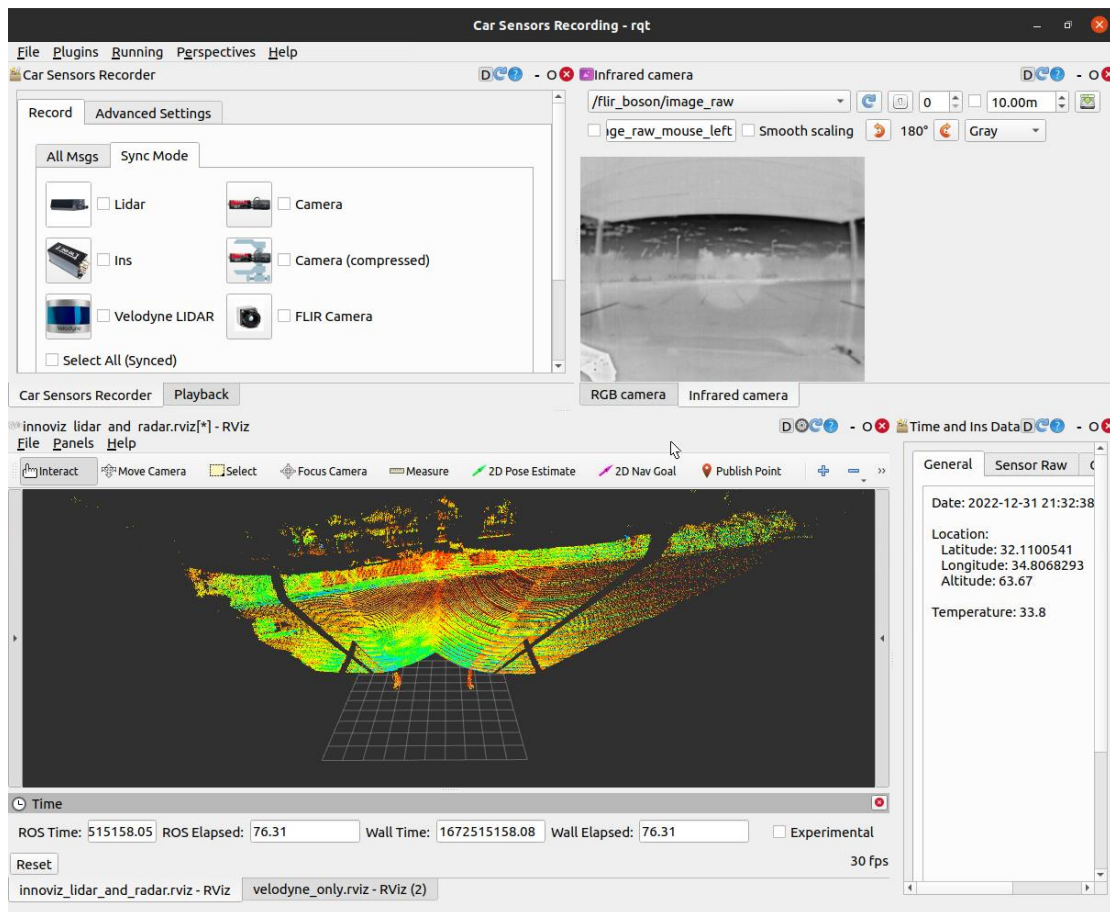
```
$ ros2 launch velodyne velodyne-all-nodes-VLP16-launch.py #not sure
```

Adding sensors to the ROS1 launch file

To add sensors to the ROS1 launch file, add nodes in the launch file. If your sensor has a launch file to run its nodes, open it and copy the contents of it to the launch file of the entire system. Note that this works only for ROS1 sensors.

Using the Recording System

The main window of the system looks like this:



The window contains the following plugins:

- Recording control plugin – top left
- Playback plugin – top left, second tab (not shown, selectable by the tab underneath)
- Camera feed plugin – top right (there are two camera feed plugins active, selectable by the tabs underneath)
- RViz plugin – bottom left (there are two RViz plugins active, selectable by tabs underneath)
- INS Data view plugin – bottom right

Recording

To select sensors to record:

1. Select normal recording (All Msgs tab) or synchronized recording (Sync Mode) in the recording control plugin.
2. Select the sensors to record using their checkboxes, or check the "select all" checkbox to select all.
3. Press the "record" button. If the record button is not visible, scroll down the plugin to access it.

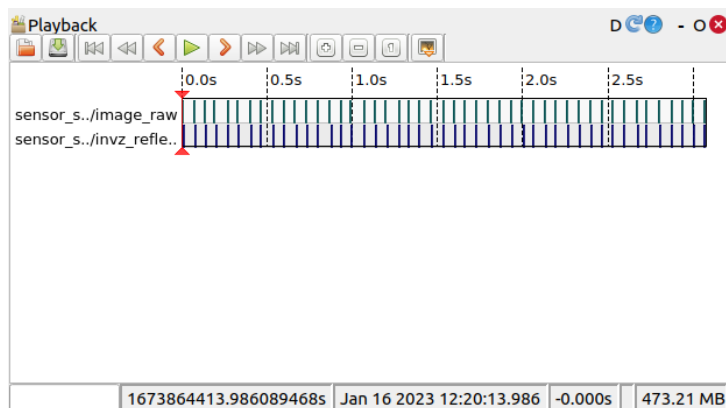
4. Stop the recording when you are done.

The recording is saved in the location set in the "Advanced Settings" tab of the recording control plugin. In our case, recordings are saved in the `~/ros` folder. The recordings are saved under a name comprised of the date and time of the recording.

Viewing a Recording

1. Switch to the "playback" plugin with the tabs under the recording control plugin.
2. Press the "open" button (with a folder icon on it)
3. Select your bagfile to view

Your bagfile is now open in the playback control plugin. It should now look like this:



Each row represents a recorded topic, each vertical line in a row represents one "frame", or message. To play the recording press the "play" button. The recording is played on the `/sensor_playback/<recorded_topic_name>` topic, so to view the recording you should switch the topics in the viewer plugins (like RViz or the camera feed viewers) to the playback topic, and return it to the previous state after finishing viewing the recording.

In RViz, if the display selection pane is hidden press on the arrow bar to the left of the viewer window. In the image view plugins, if the topic is not listed in the dropdown click the blue refresh button and then check again.