004 大末建設用ROS1Noeticの環境構築

Daisue Construction ROS1Noetic environment construction

This article describes the steps from installing Ubuntu 20.04 to saving sensor data.
Analysis will be summarized in a separate document.
The environment is as follows.

- OS:Ubuntu 20.04
- CPU: Intel n100

Here's a summary of what's being done:

- Useful settings for Ubuntu20.04
- Setting up the ROS Noetic environment
- Building an environment for realsense sensor (RGB-D camera)
- Setting up an environment for the livox mid-360 sensor (3DLiDAR)
- Create a launch file for measurement
- rosbag command for data collection

Ubuntu related settings

1. Suppression of blank screen

Power icon in the top right corner of the screen ⇒ "Settings" ⇒ "Power" in the left column ⇒ Set to "Do not blank screen"

2. Install development libraries

```
sudo apt -y install vim curl openssh-server net-tools git wget cmake
```

3. Granting read/write permissions to USB

How to grant dialout group privileges to user (change the username if it is different)

```
sudo usermod -aG dialout user
sudo shutdown -r now
```

4. Reduce the time it takes for the GRUB boot menu to display

`/boot/grub/grub.cfg`Edit
The default `set timeout=30` is . `set timeout=5` Edit it to look like this:

```
sudo nano /boot/grub/grub.cfg
```

5. Shut down when you press the power button

Open the following in an editor

```
sudo nano /etc/acpi/events/powerbtn
```

Write the following and close

```
event=button[ /]power
action=/sbin/shutdown -h now
```

Reload after writing

```
sudo service acpid restart
```

6. Hide crash reports

```
sudo sed -i 's/enabled=1/enabled=0/' /etc/default/apport
```

7. Specify DNS Search Order

Comment out (add # to the beginning) the part that says to edit /etc/nsswitch.conf `host` and rewrite it as follows:

```
sudo nano /etc/nsswitch.conf
```

```
#hosts:          files mdns4_minimal [NOTFOUND=return] dns myhostname
hosts:          files dns myhostname
```

8. Install Google Chrome

Search for "google chrome" in your browser, `.debファイル` download and save it, then run it from the software installation and install it.

Installing Google Chrome

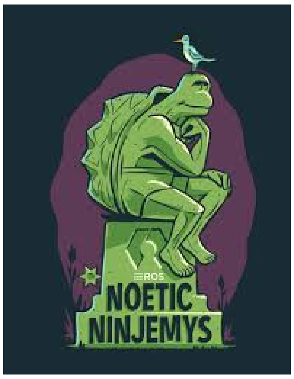
```
sudo dpkg -i debファイル名
```

9. Install Visual Studio Code

`.debファイル` Download and save the software from the following page (or search for "vscode" in your browser) . Then install it by running the software installation.

```
sudo dpkg -i debファイル名
```

Installing ROS Noetic



Add ROS to the apt 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 an apt-key to install ROS from apt

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

Installing ROS Noetic

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

Installing and initializing rosdep, which organizes ROS package dependencies

```
sudo apt install python3-rosdep
sudo rosdep init
rosdep update
```

ROS environment settings

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Install tools that make it easier to install ROS packages

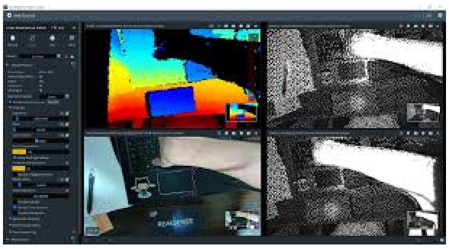
```
sudo apt install python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
sudo apt-get install python3-catkin-tools
```

Create a ROS workspace and set up your environment

The following `catkin_make` uses , but `catkin build` you can also use

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Building a realsense ROS environment



Installing the realsense library

Registering the apt repository

```
curl -sSL 'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xF6E65AC044F831AC80A06380C8B3A55A6F3EFCDE' | sudo apt-key add -
sudo add-apt-repository "deb https://librealsense.intel.com/Debian/apt-repo $(lsb_release -cs) main" -u
```

Installing librealsense

```
sudo apt update
sudo apt install librealsense2 librealsense2-dkms librealsense2-utils librealsense2-dev librealsense2-dbg
```

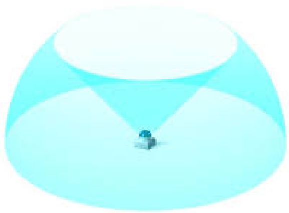
Update librealsense

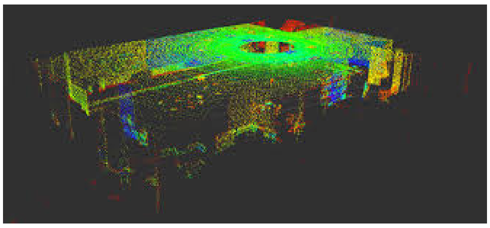
```
sudo apt update
sudo apt --only-upgrade install librealsense2-utils librealsense2-dkms librealsense2-dev librealsense2-dbg
```

Installing realsense ROS packages

```
sudo apt install ros-noetic-realsense2-description
sudo apt install ros-noetic-realsense2-camera
```

Building a ROS environment for Livox Mid360





Livox_SDK2 build and install

```
cd $HOME
git clone https://github.com/Livox-SDK/Livox-SDK2
cd $HOME/Livox-SDK2/ && mkdir build && cd build
cmake .. && make -j
sudo make install
```

Install livox_ros_driver2 ROS package

> #### Creating a workspace and git cloning

```
mkdir $HOME/catkin_ws/src -p
cd $HOME/catkin_ws/src
git clone https://github.com/Livox-SDK/livox_ros_driver2
```

> #### Rewriting CMakeLists.txt for normal build

`livox_ros_driver2パッケージ`According to the README, `build.sh`it is instructed to build with the ROS1 or ROS2 option in the file located directly under the package.

livox_ros_driver2 is shared by both ROS1 and ROS2, and the detailed specifications differ for each environment. `build.sh`If you do not use livox_ros_driver2 and run catkin_make or colcon build directly under the ROS workspace directory, the build will fail.

However, `build.sh`it says to do a clean build every time, which takes a long time to build and is troublesome because it is different from the normal build method.

`CMakeLists.txt`So, we will rewrite it so that it can be built normally .

`$HOME/mid360_ws/src/livox_ros_driver2/CMakeLists.txt`You can build it `catkin_make`with normal or by adding the following to the first line : `colcon build --symlink-install`

One thing to note is that ROS1 is written so that the build type defaults to Release mode, but this is not written on the ROS2 side, so it defaults to Debug mode. For now, I don't plan on debugging, so I've added it so that Release mode defaults when using ROS2. Below, the ROS2 settings have been commented out because this is for the ROS1 environment.

```
# ROS1の時
set(ROS_EDITION "ROS1")

# ROS2(Galactic以前)の時
#set(ROS_EDITION "ROS2")
#if(NOT CMAKE_BUILD_TYPE)
#  set(CMAKE_BUILD_TYPE "Release" CACHE STRING "Choose Release or Debug" FORCE)
#endif()

# ROS2(Humble)の時
#set(ROS_EDITION "ROS2")
#set(HUMBLE_ROS "humble")
#if(NOT CMAKE_BUILD_TYPE)
#  set(CMAKE_BUILD_TYPE "Release" CACHE STRING "Choose Release or Debug" FORCE)
#endif()
```

Duplicating package.xml

livox_ros_driver2 is compatible with ROS1/ROS2, so `package.xml`there is no , `package_ROS1.xml`and `package_ROS2.xml`are provided, and `build.sh`you can create it by copying . `build.sh`Since we will not be using it this time, we will copy it manually.

```
cd $HOME/catkin_ws/src/livox_ros_driver2/
cp package_ROS1.xml package.xml
```

Build

```
cd $HOME/catkin_ws/
catkin_make
```

Network settings for livox Mid360 (fixed wired LAN settings)

Mid-360 Quick Start Guide p.23

Livox Mid-360 supports two IP modes: Dynamic IP address mode and Static IP address mode.
All Livox Mid-360 LiDAR sensors are set by default to Static IP address mode (IP address 192.168.1.1XX),
where XX are the last two digits of the serial number of your Livox Mid-360 LiDAR sensor (the serial number can be found under the QR code next to the connector on the sensor). All
Livox Mid-360 LiDAR sensors are set by default to a subnet mask of 255.255.255.0 and a default gateway of 192.168.1.1.

Therefore, you can communicate with Livox Mid-360 by setting the PC's wired LAN settings to a fixed IP of 192.168.1.XX, a subnet mask of 255.255.255.0, and a default gateway of 192.168.1.1. (However, since this will be used with ROS1/2, the PC's IP address in the libox_ros_driver2 package is assumed to be 192.168.1.5 by default, so the PC's IP address will be set to 192.168.1.5.)

Communication check

Once the network settings are complete, `pingコマンド`check communication with. The last two digits of the serial number of the Livox Mid-360 in this case were 51, so the address is 192.168.1.151.

```
ping 192.168.1.151
```

Check that communication is established.

`livox_ros_driver2/config/MID360_config.json`Edit

- PC wired LAN: 192.168.1.5
- Livox Mid360: 192.168.1.151

In the above case, do as follows. If your IP address is different, please change it accordingly.

```
{
  "lidar_summary_info" : {
    "lidar_type": 8
  },
  "MID360": {
    "lidar_net_info" : {
      "cmd_data_port": 56100,
      "push_msg_port": 56200,
      "point_data_port": 56300,
      "imu_data_port": 56400,
      "log_data_port": 56500
    },
    "host_net_info" : {
      "cmd_data_ip" : "192.168.1.5",
      "cmd_data_port": 56101,
      "push_msg_ip": "192.168.1.5",
      "push_msg_port": 56201,
      "point_data_ip": "192.168.1.5",
      "point_data_port": 56301,
      "imu_data_ip" : "192.168.1.5",
      "imu_data_port": 56401,
      "log_data_ip" : "",
      "log_data_port": 56501
    }
  },
  "lidar_configs" : [
    {
      "ip" : "192.168.1.151",
      "pcl_data_type" : 1,
      "pattern_mode" : 0,
      "extrinsic_parameter" : {
        "roll": 0.0,
        "pitch": 0.0,
        "yaw": 0.0,
        "x": 1000,
        "y": 0,
        "z": 0
      }
    }
  ]
}
```

Source code modification for livox mid-360

livox_ros_driver2 supports two types of topic output (`livox_ros_driver2/CustomMsg` ,) for point cloud output . The topic name is . The output format can be changed by switching . When 0, it is , and when 1, it is . However, simultaneous output is not the default. is a proprietary format, and contains all the information that mid360 can output, but cannot be visualized with . Since is the standard format of ROS , it can be visualized with , but for some reason it is output in a format that loses the time information of each point cloud. In sensor data measurement, we want to visualize it and save all the data as much as possible. Therefore, we will make changes to the source code so that it can be output simultaneously. Furthermore, it is not good that it is in PointCloud2 format, which loses the time information of each point cloud, so we will fix that as well. In addition, mid360 is equipped with a 6-axis IMU, and it is output with the topic name . However, although the acceleration data is in SI units (㎡/sec) in the ROS unit system, the output is in G units, so we will make changes to output both SI units and G units simultaneously. In short, we will change the two outputs of and to the four outputs below. `snesor_msgs/PointCloud2` `/livox/lidar` `xfer_format` `snesor_msgs/PointCloud2` `livox_ros_driver2/CustomMsg`

`livox_ros_driver2/CustomMsg` `rviz` `snesor_msgs/PointCloud2` `rviz`

`/livox/imu`

`/livox/lidar` `/livox/imu`

- `/livox/lidar[livox_ros_driver2/CustomMsg]`: Point cloud format for Livox (can be used directly with Fastlio, etc.)
- `/points_raw[snesor_msgs/PointCloud2]`: ROS standard format for point clouds (can be rendered with rviz, and ring data is added for use with LIO-SAM, etc.)
- `/livox/imu[sensor_msgs/Imu]`:IMU Livox exclusive format (acceleration unit is G, can be used as is with Fastlio, etc.)
- `/imu_raw[sensor_msgs/Imu]`: IMU ROS standard format (acceleration unit is m2/sec, can be used with LIO-SAM, etc.)

*fastlio and LIO-SAM are ROS packages for LiDAR-Inertial-SLAM/Odometry (may be used during analysis).

The modified file `livox_ros_driver2/src/lddc.cpp` is. There are four modifications, all of which are additions. The number of lines is the number of lines before the changes.

- `#include "lds_lidar.h"` ①. Add the following below line 37. This is a type definition for the ROS standard format (there is no `sensor_msgs/PointCloud2` set name for it, but since it is the format used by the Velodyne sensor, it is called the Velodyne format, or the XYZIRT type) with the ring data required for calculating each point cloud time added.

```
struct PointXYZIRT
{
    PCL_ADD_POINT4D
    PCL_ADD_INTENSITY;
    uint16_t ring;
    float time;
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
} EIGEN_ALIGN16;
POINT_CLOUD_REGISTER_POINT_STRUCT (PointXYZIRT,
    (float, x, x) (float, y, y) (float, z, z) (float, intensity, intensity)
    (uint16_t, ring, ring) (float, time, time)
)
pcl::PointCloud<PointXYZIRT>::Ptr velodyne_cloud_out(new pcl::PointCloud<PointXYZIRT>());
```

- `livox_msg.points.push_back(std::move(point));` ②. Add the following to the for statement on line 392. `livox_ros_driver2/CustomMsg` This `snesor_msgs/PointCloud2` converts to (XYZIRT type).

```
    PointXYZIRT p;
    p.x = point.x;
    p.y = point.y;
    p.z = point.z;
    p.intensity = point.reflectivity;
    p.ring = point.line;
    p.time = point.offset_time / 1000000000.0;
    velodyne_cloud_out->push_back(p);
```

- `PublishCustomPointData` ③. Add the following to the end of the function on line 396. `snesor_msgs/PointCloud2` (XYZIRT type) is output.

```
  sensor_msgs::PointCloud2 cloud_temp;
  pcl::toROSMsg(*velodyne_cloud_out, cloud_temp);
  cloud_temp.header = livox_msg.header;
  static ros::Publisher pub_velodyne_cloud = cur_node_->advertise<sensor_msgs::PointCloud2> ("/points_raw", 10);
  pub_velodyne_cloud.publish(cloud_temp);
  velodyne_cloud_out->clear();
```

- `PublishImuData` ④. Add the following to the end of the function on line 493. This is the IMU output with acceleration converted to m2/s.

```
  ImuMsg imu_raw_msg;
  imu_raw_msg.header = imu_msg.header;
  imu_raw_msg.angular_velocity.x = imu_msg.angular_velocity.x;
  imu_raw_msg.angular_velocity.y = imu_msg.angular_velocity.y;
  imu_raw_msg.angular_velocity.z = imu_msg.angular_velocity.z;
  imu_raw_msg.linear_acceleration.x = imu_msg.linear_acceleration.x * 9.80665;
  imu_raw_msg.linear_acceleration.y = imu_msg.linear_acceleration.y * 9.80665;
  imu_raw_msg.linear_acceleration.z = imu_msg.linear_acceleration.z * 9.80665;
  static ros::Publisher pub_imu_raw = cur_node_->advertise<sensor_msgs::Imu> ("/imu_raw", 10);
  pub_imu_raw.publish(imu_raw_msg);
```

After making the above changes and building the build, `xfer_format` when is 1, 4 topics will be output.

Creating a launch file for measurement

You can create the launch file anywhere, but in this case we will create it below `livox_ros_driver2` in the package . `launch_ROS1` `record_sensor_data.launch`

```
cd $HOME/catkin_ws/src/livox_ros_driver2/launch_ROS1/
code record_sensor_data.launch
```

`record_sensor_data.launch`Write the following inside . If the positional relationship from the center of the bottom of the robot to each sensor is known, remove the comment out and write the relative position in `tf2_ros` within the node tag of . Write in the order of . `args` `args` `x y z yaw pitch roll 親フレーム 子フレーム`

```
<launch>

    <!-- TF -->
    <!--
    <node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_livox_frame" args=" 0.0 0.0 0.0 0.0 0.0 0.0
base_link livox_frame" />
    <node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_camera_link" args=" 0.0 0.0 0.0 0.0 0.0 0.0
base_link camera_link" />
    -->

    <!--user configure parameters for ros start-->
    <arg name="lvx_file_path" default="livox_test.lvx"/>
    <arg name="bd_list" default="100000000000000"/>
    <arg name="xfer_format" default="1"/>
    <arg name="multi_topic" default="0"/>
    <arg name="data_src" default="0"/>
    <arg name="publish_freq" default="10.0"/>
    <arg name="output_type" default="0"/>
    <arg name="rviz_enable" default="true"/>
    <arg name="rosbag_enable" default="false"/>
    <arg name="cmdline_arg" default="$(arg bd_list)"/>
    <arg name="msg_frame_id" default="livox_frame"/>
    <arg name="lidar_bag" default="true"/>
    <arg name="imu_bag" default="true"/>
    <!--user configure parameters for ros end-->
    <param name="xfer_format" value="$(arg xfer_format)"/>
    <param name="multi_topic" value="$(arg multi_topic)"/>
    <param name="data_src" value="$(arg data_src)"/>
    <param name="publish_freq" type="double" value="$(arg publish_freq)"/>
    <param name="output_data_type" value="$(arg output_type)"/>
    <param name="cmdline_str" type="string" value="$(arg bd_list)"/>
    <param name="cmdline_file_path" type="string" value="$(arg lvx_file_path)"/>
    <param name="user_config_path" type="string" value="$(find livox_ros_driver2)/config/MID360_config.json"/>
    <param name="frame_id" type="string" value="$(arg msg_frame_id)"/>
    <param name="enable_lidar_bag" type="bool" value="$(arg lidar_bag)"/>
    <param name="enable_imu_bag" type="bool" value="$(arg imu_bag)"/>

    <node name="livox_lidar_publisher2" pkg="livox_ros_driver2" type="livox_ros_driver2_node" required="true"
output="screen" args="$(arg cmdline_arg)"/>

    <!-- realsense -->
    <include file="$(find realsense2_camera)/launch/rs_camera.launch">
        <arg name="enable_infra" default="true"/>
        <arg name="enable_infra1" default="true"/>
        <arg name="enable_infra2" default="true"/>
        <arg name="enable_gyro" default="true"/>
        <arg name="enable_accel" default="true"/>
        <arg name="gyro_fps"            default="200"/>
        <arg name="accel_fps"           default="250"/>
        <arg name="unite_imu_method" value="linear_interpolation"/>
        <arg name="enable_pointcloud" default="true"/>
    </include>

    <!-- RVIZ -->
    <group if="$(arg rviz_enable)">
        <node name="livox_rviz" pkg="rviz" type="rviz" respawn="true"
                        args="-d $(find livox_ros_driver2)/config/display_point_cloud_ROS1.rviz"/>
    </group>

    <!-- ROSBAG -->
    <group if="$(arg rosbag_enable)">
        <node pkg="rosbag" type="record" name="record" output="screen"
            args="/livox/imu /livox/lidar /points_raw /imu_raw /tf /tf_static /camera/color/camera_info
/camera/color/image_raw /camera/depth/camera_info /camera/depth/color/points /camera/depth/image_rect_raw
/camera/extrinsics/depth_to_color /camera/accel/imu_info /camera/gyro/imu_info /camera/imu"/>
    </group>

</launch>
```

Build it and make sure the path is correct.

```
cd $HOME/catkin_ws/
catkin_make
source ~/.bashrc
```

To execute it, use the command below. If you change the `<arg name="rosbag_enable" default="false"/>` to in the launch file `true`, data saving will begin as soon as the program is launched. Since the name of the saved data is not specified, a file called "date and time when acquisition started.bag" will be generated in the location where the command was executed. If you want to save it under a name of your choice, you can do so by adding to within `rosbag` the node tag of . `args` `-O bag_name`

```
roslaunch livox_ros_driver2 record_sensor_data.launch
# コマンドライン上でrosbag_enableをtrueに変更するときは下記
roslaunch livox_ros_driver2 record_sensor_data.launch
```

If tab completion does not appear in roslaunch, the solution is to run the following command.

```
rospack profile
```

rosbag command for data collection (if you want to separate sensor startup and data storage)

`rosbag -a` There are too many realsense topics, so images cannot be saved properly. If you specify them individually as shown below, they can be saved properly. `-O` The following is the file name of the rosbag. If none is specified, the bag file of the system time will be saved.

```
rosbag record -O ril111eft /livox/imu /livox/lidar /points_raw /imu_raw /tf /tf_static /camera/color/camera_info
/camera/color/image_raw /camera/depth/camera_info /camera/depth/color/points /camera/depth/image_rect_raw
/camera/extrinsics/depth_to_color /camera/accel/imu_info /camera/gyro/imu_info /camera/imu
```

bonus

Installing Paint Tool

When using the autonomous driving function of ROS2, we use a paint tool to edit the 2D map.
The standard 2D map of ROS2 `.pgm` has a file extension, and we will install a paint tool that can edit it.

- GIMP: A famous multi-function painting tool (it has many functions, so it takes some practice to get the hang of it)
- kolorpaint: Intuitive to use, just like the standard Windows paint tool

```
# GIMPのインストール
sudo apt install gimp
# kolorpaintのインストール
sudo apt install kolourpaint
```