Waymo Lidar simulation (honeycomb_simulator)

Overview

The honeycomb_simulator package provides a ROS packages to simulate the Waymo lidar using Gazebo Simulator.

This includes two ros package: - **`honeycomb_description`**: Lidar's 3D model, xacro file to generate URDF, and launch files. - **`honeycomb_gazebo_plugins`**: Custom lidar simulation plugin with dynamic reconfigure support, and a sample gazebo world.

Please consider that this is an experimental tool at this point and subject to frequent changes. The **`honeycomb_on_air_stand`** is intended to serve as an example for you to copy from and place simulated lidar model on wherever you want to mount it like your robot and simulate.

Setup

Prerequisite

- Install ROS melodic or noetic (http://wiki.ros.org/Installation)
- Install Gazebo >=9.4.0 (http://gazebosim.org/tutorials?tut=install_ubuntu&ver=9.0)
- ROS workspace with package hc in it (/honeycomb/documents/ros-driver#workspace)

Build

- Extract the package **`honeycomb_simulator.tar.gz`** your workspace.

```
cd catkin_ws/src
tar -xzf <PATH_TO_FILE>/honeycomb_simulator.tar.gz
```

- Build the catkin workspace, with honeycomb_simulator in it and source it.

  ```
  cd catkin_ws
  catkin_make
  source devel/setup.bash
  ```

Simulating

To start a simulation in `gazebo` and show the `PointCloud2` in `Rviz`

```
roslaunch honeycomb_description hc_sim_in_gazebo.launch
```

Launch files

- **`rviz_only.launch`**: does not simulate, only shows xacro model in rviz, useful for editing model, for example when doing placement on a robot.
- **`hc_sim_in_gazebo.launch`**: Runs simulation in gazebo with default params
- **`example_sim_point_cloud_only.launch`** : Runs **`hc_sim_in_gazebo.launch`** with params set to publish point cloud only and not connect with driver
- **`example_sim_udp_only.launch`** : Runs **`hc_sim_in_gazebo.launch`** with params set to connect with driver only through UDP and in headless mode, with no GUI and no direct point cloud publishing

Parameters

- **`gpu`** (default true): To simulate using rendered depth image instead of ray tracing. This is generally faster, but less accurate. To run using CPU only run:

  ```
  roslaunch honeycomb_description hc_sim_in_gazebo.launch gpu:=false
  ```

- **`rviz`** (default true): Launches rviz alongside
- **`world_name`**: Set gazebo world path to run simulation in.

Topics

- **`/points_sim`** (std_msgs/PointCloud2 (http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/PointCloud2.html))

  The simulated point data as PointCloud2 message is published with XYZ position and attributes like intensity, roll, pitch, yaw values. Intensity is not simulated and preset as a channel mostly for compatibility.

- **`/imu/data_raw`** (std_msgs/Imu (http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/Imu.html))

  Simulated Inertial Measurement Unit (IMU) which matches the data rate of Honeycomb. This also is there for compatibility and does not model noise, bias or any other characteristics of honeycomb lidar.

Dynamic Parameters

It is a subset of the dynamic parameters available in hc_node, with the same functionality as mentioned in the ros driver manual (/honeycomb/documents/ros-driver#dynamic_parameters). The Dynamic Reconfigure GUI should start with the simulation launch. To start it manually, run **`rosrun rqt_reconfigure rqt_reconfigure`**

Customization using Xacro/URDF

Xacro file generates the lidar URDF model.

Include lidar xacro

You may want to include the xacro file in your project.

`honeycomb_on_air_stand.urdf.xacro` serves as an example. Where we include the lidar as

```
<xacro:include filename="$(find honeycomb_description)/urdf/Honeycomb.urdf.xac
<xacro:Honeycomb parent="base_link" name="hc" topic="/points_sim" gpu="${gpu}"
  <origin xyz="0.0 0 1.0" rpy="0 0 0" />
</xacro:Honeycomb>
```

Parameters

- `origin`: The block param, specifying translation and rotation from the parent
- `parent`: The mount point, can be some link, like the head of your robot
- `name`: Important to set different names when adding multiple lidar
- `hz`: Point cloud data output frequency
- `gpu`: Use gpu mode for simulation
- `pc_enable`: Publishing direct Point Cloud enable
- `pc_topic`: Point Cloud topic name
- `imu_topic`: IMU topic name
- `connect_to_driver` : Connect and send laser scan data to driver using UDP
- `driver_ip` : IPv4 address of the driver to connect to
- `driver_port`: Port at which the driver is listening to

Modify lidar xacro

It may be important to modify some xacro property to improve performance of simulation. Right now to allow dynamic reconfigure, the properties are set to allow maximum FoV and variety or resolutions, based on Horizontal and Vertical frequency. If you already know some this values, feel free to update them in the xacro, or a copy of it.

Properties

- `max_fov_horizontal`: Maximum horizontal field of view in degrees (default 360)

- **`min_hz_horizontal`**: Minimum horizontal Hz simulated, which is used to calculate maximum horizontal resolution (default 5). It works best when the Hz you use in dynamic reconfigure is a integer multiple of this value.
- **`min_angle_vertical`**: Minimum pitch angle in degrees (default -78)
- **`max_angle_vertical`**: Maximum pitch angle in degrees (default 20.75)
- **`max_res_vertical`**: Maximum vertical resolution in degrees (default 0.75)
- **`max_hz_vertical`**: Maximum vertical frequency (default 1500)
- **`max_range`**: Maximum range (default 50)

Using driver to connect to simulation

If you want to connect to a Simulator Gazebo plugin, you must use the Simulator connector. The connector is a drop in replacement for the shared C/C++ library.

If you set **`connect_to_driver`** to true within the plugin, the plugin will connect to the connector at the corresponding **`driver_ip`** and **`driver_port`**. When in this mode, the Gazebo plugin receives its dynamic configuration from the driver.

The Simulator connector **`honeycomb_sim_api`** contains a drop in replacement for the shared C/C++ library in **`honeycomb`** api.

The connector is beta software, and is not feature complete.

Loading sim api

Honeycomb_sim_api contains **`libhoneycomb_c_api.so`**. You can replace your original **`libhoneycomb_c_api.so`** with this one. The ideal and recommended way is to set **`LD_LIBRARY_PATH`**.

For a single run:

```
LD_LIBRARY_PATH=<path/to/libhoneycomb_c_api.so_for_sim> ./<app_to_run>
```

For setting in environment:

```
export LD_LIBRARY_PATH=<path/to/libhoneycomb_c_api.so_for_sim>:$LD_LIBRARY_PATH
```

Example run:

```
LD_LIBRARY_PATH=~/honeycomb_sim_api ./hc_scanner
```

Setting port

By default, the simulator connects to the **honeycomb_sim_c_api** at port **9090**. You may want to change that for different reasons including connecting to multiple drivers/**honeycomb_sim_c_api** simulating connection to multiple Lidars.

Port as MAC

In code you can set a port number as string in place of the MAC address. This is done deliberately to maintain compatibility with existing API.

Example with ROS node:

```
<node name="hc1" pkg="hc" type="hc_node">
    <param name="hostname" value="127.0.0.1"/>
    <param name="mac_address" type="string" value="9090"/>
    <param name="frame_id" value="hc1"/>
    <remap from="points" to="points1" />
</node>
<node name="hc2" pkg="hc" type="hc_node">
    <param name="hostname" value="127.0.0.1"/>
    <param name="mac_address" type="string" value="9091"/>
    <param name="frame_id" value="hc2"/>
    <remap from="points" to="points2" />
</node>
```

Example in C++:

```
waymo::Honeycomb lidar;

if ((status = lidar.Init("","9092")) != waymo::Status::kOk){
  std::cerr << "Failed to initialize Honeycomb lidar: " << status << std::endl;
  return 1;
}
```

Port in Env

You can also override the port by setting environment variable `HC_SIM_PORT`.

Example roslaunch:

```
HC_SIM_PORT=9091 roslaunch hc hc.launch
```

Example hc_scanner:

```
HC_SIM_PORT=9091 hc_scanner
```

Example in launch file:

```
<env name="HC_SIM_PORT" value="9191" />
<node name="hc" pkg="hc" type="hc_node" output="screen" />
<env name="HC_SIM_PORT" value="9192" />
<node name="hc2" pkg="hc" type="hc_node" output="screen" />
```

Troubleshooting

If you have loaded the correct library, upon scanning you would see something like **`HoneycombSIM: Scanner at port 9090`** and for Init **`HoneycombSIM: Init driver at port 9090`**.

If you don't see `HoneycombSIM:`, make sure that path to library is not wrong, and print is not suppressed/sent to log file.

License

Proprietary

Last updated 2021-04-08 UTC.