Università di Pisa

Smart Application

A.Y. 2021/2022

Sensory ROS Nodes Documentation

Amico, Dalla Noce, Telila

Contents

1 ROS nodes team task

A node represents a process that performs some computation by receiving input from other topics and publishing a result to a desired topic. We have created 4 nodes that perform cone detection in both stereo camera and lidar sensor, one node for output fusion to perform and human interaction to interact with a car during or before the start of the competition.

2 General architecture

The below diagram in Figure 1 shows the architecture of our nodes and how they are interconnected. The nodes are represented by a circle. The arrow to a node represents the topics that the node subscribes to and the arrow out of the node represents the topic published by a node.

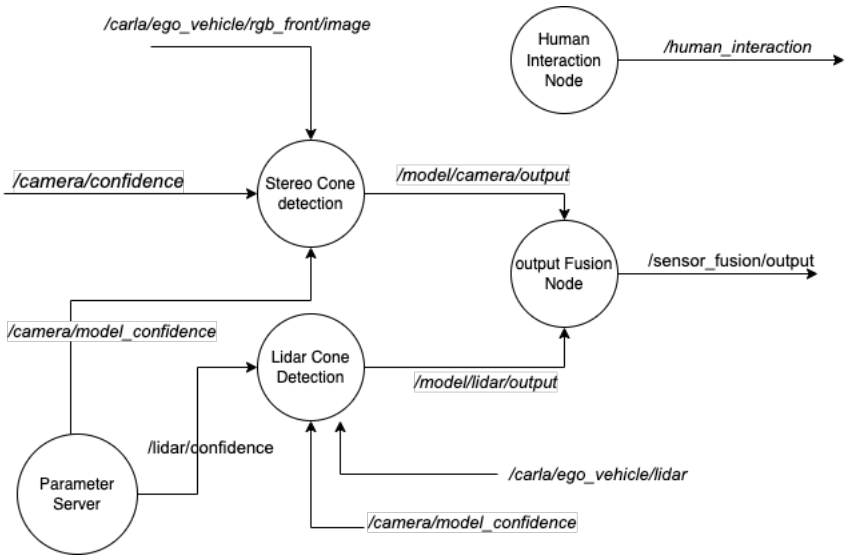

Figure 1: ROS nodes schema

3 ROS Nodes

In this section we descibe how we implemented the ROS nodes. We performed all our implementation and experiment on laptop pc with NVIDIA RTX 2060, Ryzen 7 4800H running Ubuntu 20.04 LTS.

3.1 RGB Camera cone detection node

The code is implemented in the file Rgb_camera.py. This node is responsible for recognizing cones inside the images coming from the rgb front camera. To

do that we used a Pytorch implementation of Yolo v5, in particular Yolo v5s, loaded from the pytorch hub with pre-learned weights and then fine-tuned for this specific task. In our implementation, the model can be used without an internet connection. In order to run, the model requires CUDA driver and, as a consequence, an NVIDIA GPU.

The confidence of the model can be set on the parameter server as a "/camera/model_confidence" parameter and the node retrieve it as soon as it starts running. The node tries to read this variable. If it is set, the model will use this value as a confidence, otherwise it will use the default one which is 0.74. This attribute can be adjusted while the ROS node is running by simply publishing the new desired value to the "/camera/confidence" topic.

The model gives the position of the cones as an output and the performances of the inference phase are good enough to match the frame rate of the input source. This has been tested for an input data frame rate of 20 fps, coming from the stereo camera emulated on the simulator.

This node subscribes the following topics:

- "/carla/ego_vehicle/rgb_front/image" to receive raw rgb image from the vehicle front camera. The expected type is the Image type implemented in ROS.
- "/camera/confidence": it is necessary to change the model confidence at runtime. Smaller this value is, more cones you can see, but more false positives you get. It expects a Float32 value.

And publishes to the following topic:

- "/model/camera/output" : The output from the rgb code detection model is published on this topic. It is a Float32MultiArray within which there are the positions of the cones. Each entry is formatted as [xstart, xend, ystart, yend, class]. The first four data describes the position and the dimension of the box that wraps the recognized cone, while the 'class' indicates the type of cone: 0 stands for big orange cone, 1 for small orange, 2 for yellow cones and 3 for the blue ones. This node has not a limit in the throughput and in our experiments it can carry out the same amount of data that it receives from the stereo camera, so it can operates at least at 20 Hz.

Workflow As soon as the node starts running, it loads the model that deals with the cone recognition, then it subscribes to the aforementioned topics. Every time a frame comes from the stereo camera, it goes as input to the model that detects the cones inside of it. In the end, the positions of the cones are published to the /model/camera/output topic. In the Figure 2 below you can see the model recognizing the cones that are on the street.

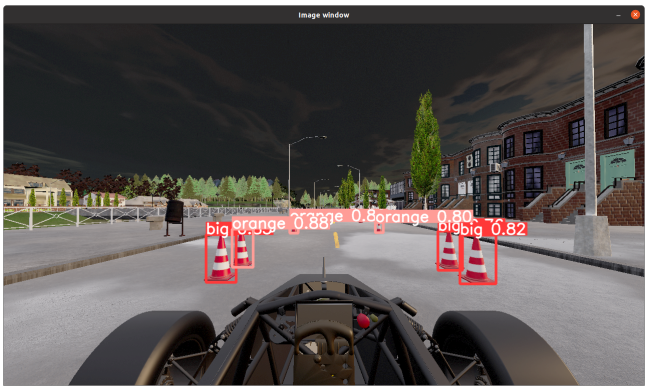

Figure 2: A visualization of the output carried out by the RGB camera cone detection node.

3.2 Lidar cone detection node

The cone detection node is implemented in the file called lidar.py. The lidar cone detection node is pretty similar to the RGB camera cone detection node described above in 3.1. This ROS node also exploit Yolo v5s to recognize the cones, but the images provided by the lidar sensor are pointclouds, much different from an RGB image. To work with this kind of data it was necessary to do another fine tuning processes. Also this model works with CUDA, and it is possible to let the two nodes run on different GPUs to increase the overall performance. Like we did before for the stereo camera node cone detection, a parameter called "/camera/model_confidence" should be set on the parameter server. If it is set, the model will use this value as a confidence, otherwise it uses a default one which is 0.4. This node subscribes to the following topics:

- "/carla/ego_vehicle/lidar" to receive the PointCloud from the lidar sensor.
- "/lidar/confidence" to change the model confidence at runtime. Smaller this value is, more cones you can see, but more false positives you get. It expects a Float32 value.

And publishes to the following topic:

- "/model/lidar/output": the node publishes a Float32MultiArray that describes the positions of cones. In particular we have an entry for each one of the nodes recognized by the model, encoded by the following format: $\langle x, y, z \rangle$, where x and y indicate the center of the cones, while the z is the distance. The origin of the axis is placed on the sensor which is centered on the car. A negative value of y means that the cone is on the left side, a positive value on the right side. About the x, higher its absolute value is, more a cone is far from the car.
  In our tests, we discovered that this node was not a bottleneck, in fact

the output rate is the same as this node has in input, 20 Hz. We did not insert any throughput limitation, so at least it can send data at 20 Hz.

Workflow When the node starts running, it loads the model that deals with the cone recognition, then it subscribes to the aforementioned topics. Every time a frame comes from the lidar, it goes as input to the model that detects the cones inside of it. In the end, the positions of the cones are published to the /model/lidar/output topic. Right below in Figure 3 you can see the model that recognizes the cones on a pointcloud provided by the lidar sensor.

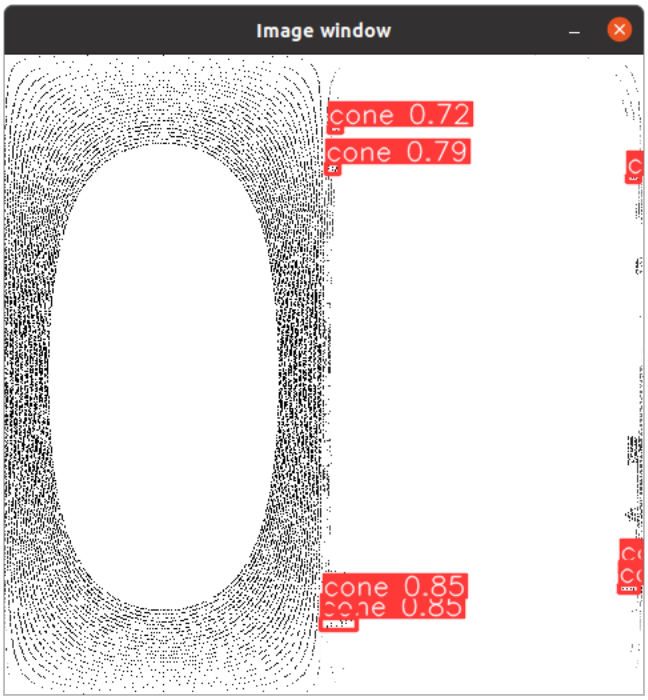

Figure 3: A visualization of the output carried out by the lidar cone detection node.

3.3 Output Fusion Node

The aim of this node is to combine the data produced by the aforementioned nodes: the lidar cone detection node and the RGB camera cone detection node. In fact, from the first node it's possible to obtain the position of the cone in the 3D space, while from the second node we can get the color and the type of the cones. This node combines those data to obtain the 3D location and the color of any detected cone. This node is subscribed to the following topics:

- "/model/lidar/output" to retrieve the 3D cone positions carried out by the lidar cone detection node.

- "/model/camera/output" to read the outputs coming from the camera cone detection.

And publishes to the following topic:

- "/sensor_fusion/output": here we publish the 3D position of the cones and their type. The data are encoded as Float32MultiArray, where each entry has this format: [x,y,z,c], where the first three variables are the one coming from the lidar cone detection node, and 'c' is the cone type, provided by the stereo cone detection node, as shown in Figure 4. Usually, the stereo camera model can recognize a major quantity of cones with respect to the lidar one. As a consequence, this node can output data only in case of a cone is recognized by both the before mentioned models. Whether any of the two models can't recognize a cone, means that either the 3D position are missing or the cone type, so the output would not be consistent.

Workflow This node works only after the rgb_camera and lidar are already running, because it uses their output to combine the recognized cones to carry out their 3D coordinates and type and published them into the "/sensor_fusion/output" topic.

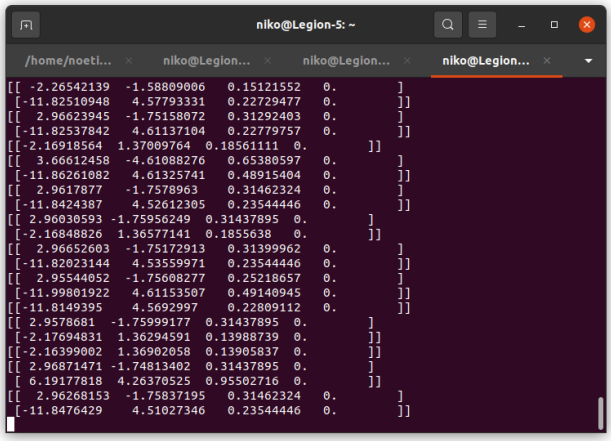

Figure 4: Sensor Fusion node output

3.4 Human Interaction Node

This node emulates a controller that the team must use when a team want to join to a competition. It is useful to send the "start" command to let the autonomous car start a competition or to make an emergency stop in case of emergency. This node publishes to the following topics:

- "/human_interaction": here we publish the argument of the script, as a String. The argument represents a command that could be either "stop" or "start". For example, *human_interaction.py start*.

3.5 Parameter server

The parameters that you can set in the parameter server for the lidar and camera models are the following.

- `/lidar/confidence std_msgs/Float32`
- `/camera/confidence std_msgs/Float32`

The value you insert indicates the confidence of the models: lower it is, higher the false positive are and vice-versa. This values are loaded as soon as the stereo camera cone detection node and the lidar cone detection node start running.

4 Let the sensors run

Run the following commands in no particular order:

- To run the **stereo camera** type: `rosrun sensory rgb_camera.py`.
- To run the **lidar** type: `rosrun sensory lidar.py`

For both the stereo camera and the lidar you can use `--visualize` flag to show the how the cone detection is working. Only after you've run both previous commands, you can run the following one for the sensor fusion, otherwise it won't work correctly.

- To run the **sensor fusion**, type: *rosrun sensory output_fusion.py*

If you want to visualize the real-time cone recognition, as for the lidar and stereo-camera, use `--visualize` at the end of the aforementioned command. The sensor fusion won't show the same graphical things as lidar and the stereo-camera do, but it will print every cone that has been detected and labeled.

4.1 Human interaction

As you know from the rules of the race, the car can start running only after it receives a "start" command. The following commands publish a string in the topic named "/human_interaction" whose value can be either "start" or "stop". The former lets the car start, while the latter suddenly stops the car in case of emergency.

- To let the car run, type: `rosrun sensory human_interaction.py start`
- In case you want to stop the car: `rosrun sensory human_interaction.py stop`

4.2 Change the confidence of the models

To change the confidence of the lidar model: "rostopic pub /lidar/confidence std_msgs/Float32 $\langle float\_value \rangle$" To change the confidence of the camera model: "rostopic pub /camera/confidence std_msgs/Float32 $\langle float\_value \rangle$".
The $\langle float\_value \rangle$ should be within the range (0, 1).