ILR - 10

Perception System Using Stereo Vision and Radar

Team A - Amit Agarwal

Harry Golash, Yihao Qian, Menghan Zhang, Zihao (Theo) Zhang

Sponsored by: Delphi Automotive

April 6, 2017

Table of Contents

1. Individual Progress 1
    1.1 Radar Setup in ROS 1
    1.2 Radar Visualization 2
2. Challenges 4
    2.1 Data Filtering 4
    2.2 Test Car 4
3. Teamwork 4
4. Future Plans 4
5. References 4

1. Individual Progress

Since the previous progress review, I was tasked with working on the Radar and visualizing the tracking points received by the Radar through Ethernet. For this I worked with Harry and obtained data about tracking points from the Radar and visualized them in RViz.

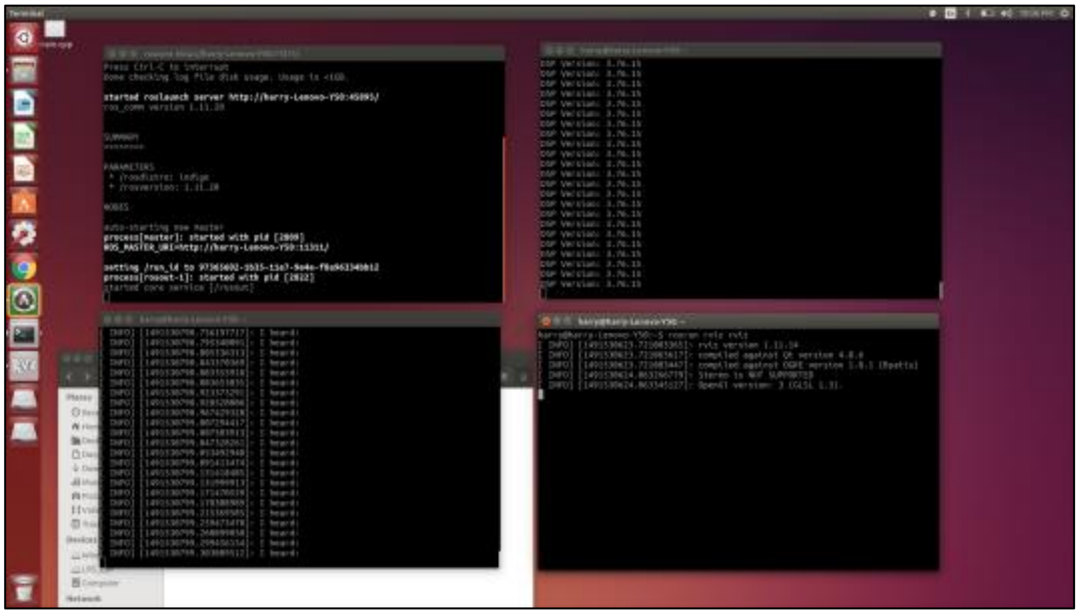
1.1 Radar Setup in ROS

Over the past couple of weeks, Harry and I have been working on getting sensible data from the Delphi ESR 2.5 Radar. The usual procedure, as described in the previous ILRs, is used to connect the Radar to the test car.

I started off by reviewing ROS documentation, since I hadn't used ROS since the assignments for the project course last semester. I reviewed major components of ROS by understanding what a topic, a node, a package, a message and a service is. In addition, I studied how a publisher subscriber works to transfer data to other nodes. While reading and experimenting with ROS, I was evaluating the pros and cons of using passing data through a message as a publisher-subscriber versus requesting the data through a service setup in ROS. Using a basic publisher-subscriber model, the nodes would be able to send and receive data continuously at the speed it is sent by the Radar. On the other hand, using a service to request the data from the node, the data can be received at the intended time intervals. Using a service there will be no wasteful data received which will not be processed by the system. Even after having these benefits with using a service, I decided to go forward with the publisher-subscriber as the team plans to filter the data before using it with the visualization. Currently, no form of filtering is applied before visualizing the data received but that will change in the coming weeks.

After a basic "Hello world" publisher subscriber was set up, the next step was to integrate the current C++ code, which would output the Radar data into terminal, with ROS so that the data would be published to a topic, which could, eventually, be subscribed to by any node in ROS. Most of the code in the cpp file did not need to be modified to work with ROS. Most of the time was taken in getting the dependencies for the C++ code to work with ROS. The major dependency that caused issues was the dependency for Boost libraries. This was not expected since Boost libraries are often used with ROS and it's used very often in similar applications. After scouring Stack Overflow and ROS forums for hours, the libraries were working and the data was getting published to the "radar" topic. The issue was mostly related to ROS not being able to find the path to the Boost libraries.

The code was initially written to transfer data for only one tracking point instead of the complete 64 points. This was done so that a basic visualization can be set up quickly. The visualization part is described in detail in section 1.2. Once one tracking point was visualized, the code was modified to instead send data about 64 tracking points. This involved modifying the parsing code for the Radar to parse information about 64 tracking points instead of just the first one. This was done using the knowledge of TCP/IP protocols and socket programming learnt before the previous progress review. Then the custom message that was being published to the “radar” topic was modified to hold an array of 64 data points for each information element like range, range rate and angle. Similarly, the rest of the code was modified to receive 64 points instead of just one point.



```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://harry-Lenovo-Y50:[illegible]/
ros_comm version 1.11.[illegible]

SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.[illegible]

NODES

auto-starting new master
process[master]: started with pid [[illegible]]
ROS_MASTER_URI=http://harry-Lenovo-Y50:11311/

setting /run_id to [illegible]
process[rosout-1]: started with pid [[illegible]]
started core service [/rosout]
```

```
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
DSP Version: 3.76.15
```

```
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
[INFO] [[illegible]]: I heard:
```

```
harry@harry-Lenovo-Y50:~$ rosrun rviz rviz
[ INFO] [[illegible]]: rviz version 1.11.[illegible]
[ INFO] [[illegible]]: compiled against Qt version 4.8.[illegible]
[ INFO] [[illegible]]: compiled against OGRE version 1.8.1 (Byatis)
[ INFO] [[illegible]]: Stereo is NOT SUPPORTED
[ INFO] [[illegible]]: OpenGl version: 3 (GLSL 1.3).
```

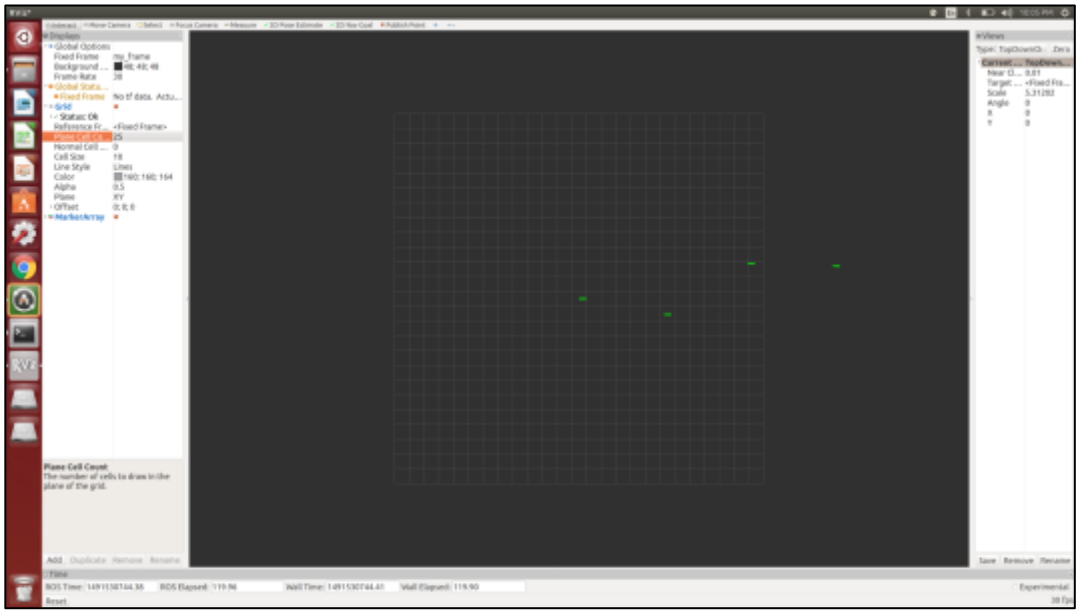
Figure 1: ROS Setup

To run the visualization a certain number of steps were followed. First, the network manager was stopped using the terminal. Then the ifconfig was modified through terminal to remap the eth0 port to the Radar. Then, roscore was started in the same terminal (Figure 1, top left). Once, roscore was up and running, in a new terminal window (Figure 1, top right), the Radar data parser was initialized using rosrun. Once, data was being received by the Radar, in a new terminal (Figure 1, bottom left), the code to format the data to Cartesian coordinates was initialized using rosrun. Finally, when the data was being published by this piece of code, in a new terminal (Figure 1, bottom right), RViz was initialized using rosrun to display the visualization (Figure 2 on the next page).

1.2 Radar Visualization

The Radar was visualized in RViz as recommended by Sean. Previously, other avenues, one of them being OpenCV, were explored for visualizing the data. After careful consideration and comparison, RViz was the better choice since the system integration was performed in ROS.

To visualize the data, a node was created to convert and format the data received from the Radar to a format suited for RViz. This mainly consisted of changing the data from Polar coordinates (range and angle) to Cartesian coordinates (x and y). The major difficulty faced in writing this code was due to lack of experience and resources on how to write a publisher as well as a subscriber in the same file. Eventually, this was done by writing the subscriber in the main function and the writing the publisher in the callback function for the subscriber. The subscriber in this file gathered data from the “radar” topic and passed that into the callback function. The callback function then converted and formatted the data to be published to the RViz topic.



Global Options	
Fixed Frame	my_frame
Background ...	48; 48; 48
Frame Rate	30
Global Statu...	
Fixed Frame	No tf data. Actu...
Grid	
Status: Ok	
Reference Fr...	<Fixed Frame>
Plane Cell Co...	25
Normal Cell ...	0
Cell Size	10
Line Style	Lines
Color	160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
MarkerArray	

Plane Cell Count:
The number of cells to draw in the plane of the grid.

Type: TopDownOr... Zero

Current ...	TopDown...
Near Cl...	0.01
Target ...	<Fixed Fra...
Scale	5.31282
Angle	0
X	0
Y	0

ROS Time: 1491530744.38 ROS Elapsed: 119.96 Wall Time: 1491530744.41 Wall Elapsed: 119.90

Experimental

30 fps

Figure 2: RViz Visualization for the Radar

Initially, the message type used to visualize the data was a marker since only one tracking point was being visualized. The shape type used to visualize the tracking points was an arrow. Currently, the length of the arrow did not signify anything. Although, an arrow was chosen so that eventually the length of the arrow would represent the range rate or the velocity of each tracking point in the visualization. This has not been done yet as a certain level of road testing is required to understand the scaling needed for this.

At first, the visualization generated was not live and was not updating as the data points arrived. There were multiple things that were changed in the publisher-subscriber so that the arrow would keep getting updated every time a new data point arrived. Once this was achieved, the code was modified to accept an array of information about all 64 tracking points. After that, the publisher to RViz was modified to send a MarkerArray instead of a Marker to allow the 64 points to be visualized live instead of just one.

2. Challenges

The major challenges faced were due to minimal previous knowledge of algorithms for data filtering from the Radar. In addition, the test car for the project has broken down.

2.1 Data Filtering

The knowledge possessed about filtering the Radar has been minimal up until this point. The significance of proper methods of filtering was realized when the visualization received from the Radar was not of the expected standard. Delphi will be helping the team in achieving this goal.

2.2 Test Car

The test car used by the team, a Volvo S60 2.5T AWD, is not in working condition anymore because of hitting an animal on the interstate over spring break. The car is undergoing repairs and should be available for testing hopefully sometime this week, so that further progress can be made.

3. Teamwork

The work this week was performed well by all 5 team members. Harry and I worked on the Radar and getting data from the Radar and visualizing the data in real-time. Yihao worked on implementing the stereo vision to work on the GPU instead of the CPU so that better frame rates can be achieved. Menghan and Zihao worked on the ROS module which will be used to integrate the Stereo Vision and Radar systems. Everyone worked well and in a timely manner.

4. Future Plans

In the future, we plan to complete system integration and move forward with validating the experiments in our test plan. Once this has been achieved, the team will proceed to prepare for SVE.

5. References

[1] "Delphi ESR 2.5". Delphi Support Center. N.p., 2017. Web. 12 Feb. 2017.
[2] "TCP/IP Network Programming". Vichargrave.github.io. N.p., 2017. Web. 24 Mar. 2017.