Individual Lab Report #9

Mar. 23, 2017

Menghan Zhang

TeamA

Amit Agarwal

Harry Golash

Yihao Qian

Zihao Zhang

Individual progress

This time, our team mainly focused on settling down the system integration method. At first we thought about build the platform on Linux by ourselves. Until now, when we formally started to integrate the system, we found that we did not know how to start.

I did some research on system integration, and compare the advantages and disadvantages of them, then we decide to use ROS. Here's some methods or platforms I have thought about.

1. Orocos Real-Time Toolkit (RTT)

   The Orocos Real-Time Toolkit (RTT) provides a C++ framework, or "runtime", targeting the implementation of (real-time and non-real-time) control systems.

   This toolkit is similar to ROS, but the mainly difference is that it focuses on the low level communication and real-time control. It can seamlessly concatenate to ROS and compared to ROS, it can achieve real time.

   The advantage is that it can:
   - Control devices ranging from sensors to complete robots
   - Capture and plot the data flows between components
   - Tune your algorithms at run-time
   - Write your controller as a hierarchical state machine
   - Configure components and the application from XML files
   - Interact with your devices directly from a GUI or command prompt
   - Extend it with your own data types
   - Extend your legacy control applications with all the above
   - Run it on standard operating systems as well as dedicated real-time systems

   The disadvantage is that:
   - Do not have large users like ROS so that it's harder to find information about that. And the documentations seem not consummate
   - We're not familiar with this toolkit and might need lots of time to get familiar with it

2. Build platform on Linux.

   The system structure shows in Figure 1. Inputs are from 3 sensors, stereo camera, Radar and GPS. Outputs are shown in a GUI I introduced last time. We were think about correlating the acquired data at the same time stamp then do the further processing. Based on the Radar situation, we probably will fuse the radar and camera at high level and give back to GUI to show. The stereo camera part basically has three parts, stereo vision, object detection, object tracking. After capturing image from cameras, the detection subsystem will give the class and location information of objects and stereo vision will give the depth information in FOV. And tracking subsystem will receive the position information of the objects and start to tracking them several frames. After several frames, the detection will run another time and this time, we can compare the results from both detection and tracking and correlate the same objects then calculate the velocity of them and add new objects on tracking and delete the disappeared objects. After that we can get relative position and velocity information of the objects then can confirm with radar data and give back to GUI to show. So the main problem is the communication and synchronization between subsystems and sensors. We can use lock free to keep the data synchronized and TCP to communicate between subsystems. It can achieve real-time, but the mainly problem is none of our teammates has CS background, and we don't have experience to build the platform also do not familiar with TCP or multi-thread coding. It must be a risk to try to build platform by ourselves.

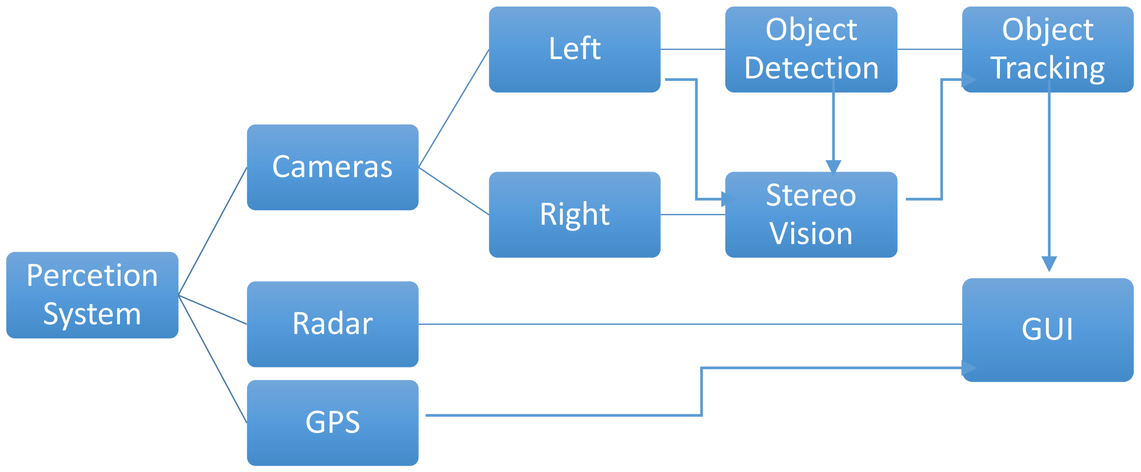

Figure 1. System Structure

3. ROS

The reason why we haven't thought about using ROS is because we worried about its real-time performances. ROS is not real-time because it is built on top of Linux, which is not inherently real-time.

Some of the reasons that ROS does that can violate real-time constraints are:

- Using a network-based transport:
- Ethernet and IP are unreliable, but have minimal transport latency.
- Standard switches and routers introduce a non-deterministic amount of latency, particularly when there is other traffic on the network.
- The handshake and retry mechanisms that make TCP reliable also introduce a significant amount of latency.
- Many of the ROS message types and other APIs do memory allocation internally. In a Linux system, this can cause a context switch which has the potential to break real-time.
- The ROS message queue will drop messages when they become full.

But we have consult with some autonomous driving company and most of them are using ROS to prototype and seem like it can achieve the real-time requirement. And ROS do have significate advantages to build a distribute systems.

- Easy to communicate between subsystems
- All of the nodes contact with ROS master which is easier to process data and synchronize them.
- Useful filesystem
- Have plenty of users and resource that we can use.

The ROS system structure will roughly looks like Figure 2.

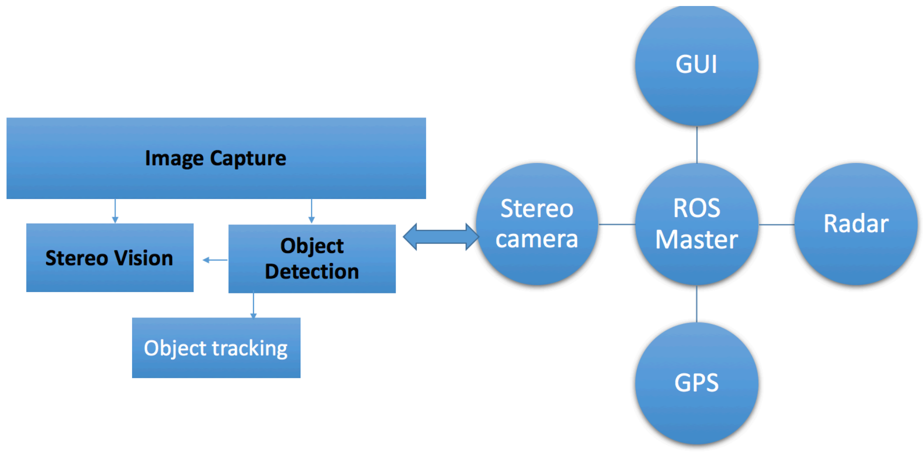

Figure 2. ROS Node Structure

4. Modify SSD to ROS node
   Since we have decided to use ROS, I started to modify Caffe and SSD into ROS node. The node should subscribe to the image_raw data captured by camera and published two topics, one is the detection results of Car another is for pedestrians. And I need 3 functions, one is to receive parameters such as pretrain models, image data and some other configuration parameters. Another one is callback which need to do detection and publish the results. Last one is used to convert the format of the messages.
   I have built a package and wrote launch file and node cpp but there are still several configuration problems.
   First thing I have done is installing and configuring the environment of Caffe into ROS catkin workspace. And then make caffe package distributed so that other packages can use all the layers.hpp in caffe distributed. Then catkin make the work space to generate bin file.

Challenges

The configuration problem is the main challenge and these problems took me several days to solve. I realized the reason might be my debug ability needs to be improved.

First one, make distribute version of Caffe. Basically modified the configuration file, CmakeLists, and Makefile, but since I barely use Cmake and make before, I need times to search online. Finally luckily solved that.

Another issue is I can not catkin make after I upgrade the systems, I tried several methods even reinstalled ROS can not solved that. Thanks Nima to help me out, the main problem is in python config file, the format need to be change back to the original version.

Teamwork

This time, Yihao initialized a multi-objects tracking algorithm. Zihao was working on change stereo vision to ROS node, Amit and Harry was keep trying to parse data from Radar and visualize it. Due to the sprint break, the team do not got chance to meet, basically Zihao, Yihao and me discussed about the system integration method then we do the tasks separately。

Future work

1. Make SSD work in ROS
2. Since we have moved to ROS, I'll try to use Rviz to visualize the final output.