Fly Sense

Shivang Baveja

Team C: FlySense

Teammates: Nihar Tadichetty, Joao Fonseca, Harikrishnan Suresh, Nicholas Crispie

ILR 04

November 10, 2017

Individual Progress

In the past two weeks, I have been able to make considerable progress with the onboard computer software. There were multiple aspects of the work done which are detailed as follows:

1. Onboard software architecture:

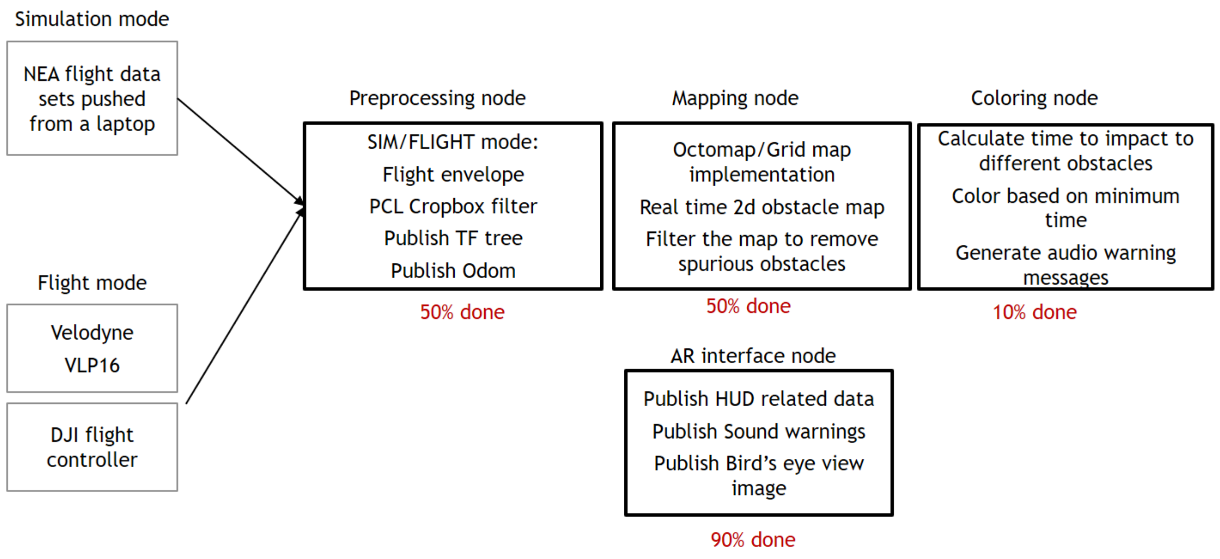

Figure 1 Onboard Software Architecture

The onboard software architecture is shown in figure 1. It was developed by keeping the following goals in mind:

a. Modularity:
   Since the onboard software will later be put on the NEA's flight system. It is important to have an interface which allows receiving data generated in NEA's flight system or from our flight hardware. Also, to avoid testing with the flight hardware every time we need to test a sub-system, it is important to be able to receive recorded data from earlier flights. These flight data could be from NEA flight system or on our flight system.

   It was also important to keep the AR interface node separate so that there's no change later in this node. Any change in our custom communication protocol requires change only in that node.

b. Setup a framework where software developed by other team members can fit in.
   The mapping node is responsible for generating the obstacle map and publishing it, which is further processed by the coloring node. Coloring node is where the obstacles in the bird's eye view image are colored based on time to impact and relevant audio warning messages are generated.

2. Preprocessing node

   The preprocessing node is responsible for following:

   a. Receive argument “flight mode”, which can either be flight mode or simulation mode.
   b. If in sim mode, use the topics published by playing the recorded flight data from a rosbag file.

      If in flight mode, start Velodyne driver which published the point cloud. Also start the DJI_SDK driver, which interfaces with the DJI Matric 100 flight controller.
   c. Calculate the flight envelope based on telemetry data received from flight controller (attitude, altitude, and speed, pilot inputs).
   d. Filter the point cloud to keep only the points in the flight envelope.
   e. Publish TF tree for mapping node.
   f. Publish odometry for the mapping node.
   g. Publish telemetry for the AR interface node.

3. Filter point cloud based on Flight Envelope:

   It is important to filter the point cloud data if we wanted to accomplish obstacle map generation onboard the quadcopter. To accomplish it there are two aspects to it which are described now:

   a. Flight envelope computation: We realized that it is difficult to precisely model the dynamics of the quadcopter, but we still wanted a way to keep only relevant points among 300,000 points that are received every second. For flight envelope, we wanted to compute the addressable area surrounding the aircraft where aircraft can reach given max pilot input commands. This simplified the problem as we don’t need to understand copter dynamics at any instant but only the final reach in each amount of time. The problem was further simplified as the DJI flight controller limits the pilot’s inputs and speed of the copter.

      Joao has been able to develop the first version of dynamics model based on translation which will later be extended to include rotation. I have been assisting him in defining the problem. It was important that the flight envelope is a cuboid and not an ellipsoid for the purpose of simplifying filtering process in software. I gave him an idea of quadcopter flight modes and what all information is available from the flight controller and what is not available.

   b. Filter point cloud: I did some research to find efficient ways to filter the point cloud and came across cropbox filter from Point cloud library. The ROS implementation for this filter wasn’t documented which took some time to figure out. Another important aspect while using this filter was to be able to dynamically configure the xmin, xmax, ymin, ymax, zmin and zmax parameters of the filter so that we can filter the point cloud based on flight envelope which is computed at every instant. The dynamic reconfiguration was tested by changing the box size continuously and observing the results in RVIZ. Figure 2 and Figure 3 shows the full point cloud and a filtered version respectively.

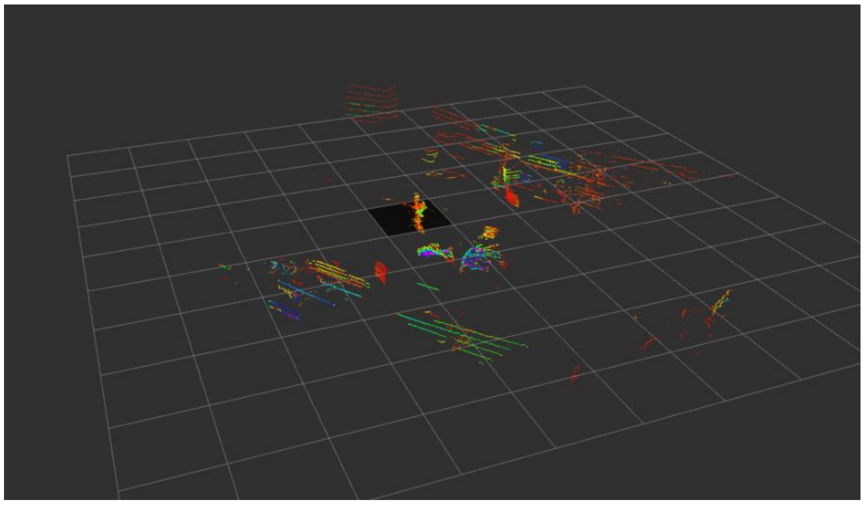

Figure 2 Full Point Cloud from VLP16

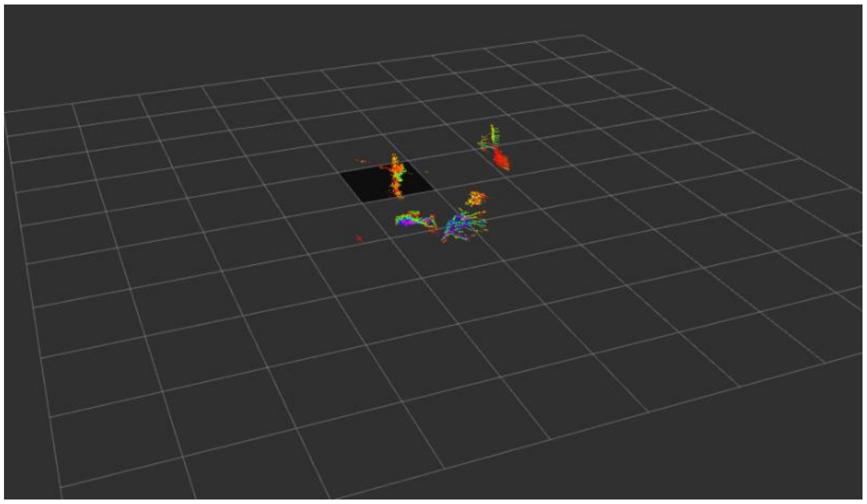

Figure 3 Filtered Point Cloud

4. DJI Flight controller interface:

   DJI provides an Onboard ROS SDK which was used to interface with the flight controller via UART serial port. The UART_Can2 port on the DJI Matrice 100 was connected to Linux machine via USB to TTL cable and a custom DJI cable.

   Following steps were involved in this process:

1. Register as a developer with DJI, to generate a key and app ID which is later used for application activation.
2. Configure the flight controller to enable onboard SDK API, set baud rate and telemetry rates.
3. Clone the DJI onboard ROS SDK from GitHub and build.
4. Modify the ROS launch file for the DJI SDK with app settings and activate the key. For the activation process, it is required to have a phone running DJI go app and connected to the DJI flight controller.
5. Run the launch file and check the published telemetry topics.

So far, the initial interface to receive data from flight controller is complete. In the coming weeks, this data will be processed to publish the tf tree, Odom and some other flight-related information.

5. AR interface node

Before writing the AR interface node, the interface specification document was written and revised so that there is no ambiguity. It was also identified that with ROS Java it is difficult to subscribe to topics publishing custom messages. So, the whole interface has been defined in a way to publish one float variable on one topic. The bird's eye view image is published as a standard camera/image type of sensor/msgs message in ROS.

The interface was tested with dummy data sent to AR interface and displayed in the headset. The rate at which data was sent was varied between 50Hz to 1Hz to see if the AR device can refresh data at that rate and what rate is good enough for the user to not get distracted by fast changing numbers. It was observed that 10Hz is a decent enough rate for the user and AR device was able to render it without any perceptible lag.

For communication, a wifi hotspot was created to which both the systems were connected. There wasn't any issue with data communication via ROS except the USB wifi dongle on Jetson behaving erratically at times. We are procuring another wifi adapter to fix this issue.

Challenges faced

- There was no documentation available to implement PCL Cropbox filter in ROS. It took some time to that out from their library documentation.
  Also, there is no specific documentation of how to dynamically configure the parameters from the node and thus was a little challenging to implement.

- Getting wifi working on Jetson TK1 was a little bit of an issue. It should be a trivial thing to do but since there's no inbuilt support for wifi adapters, right drivers have to be installed and configured to get it to work.
- Ground testing of systems in the cold 😊

Teamwork

Name	Contribution
Nihar Tadichetty	• Improving Heads-up-display in Epson AR headset, by adding vertical scrolling numbers for altitude and speed. • Implemented communication protocol on ROS Java. • Identified libraries for Audio warnings.
Joao Fonseca Reis	• Flight envelope computation based on quadcopter dynamics. He developed a model which will be used at every time instant to filter the point cloud. • Developed an initial strategy to segment obstacles into red (least time to impact), yellow(more time to impact). • Interfaced Google voice API for voice commands and conducted tests to check feasibility for our system.
Harikrishnan Suresh	• Tested the limitations of the octomap library and found that the update rate is slow for the purpose. • Implemented Grid map library to develop a clean version of the obstacle map. It's still in development phase. • Tested the above mentioned two libraries with NEA flight data sets.
Nicholas Crispie	• Detailed design of Power distribution board, developing BOM and Gerber files for PCB fabrication • Developed CAD model of the FVE hardware setup. • Project management (work-packages, procurement, schedule) • Ground testing of DJI matrice to check sensors/GPS

Plans

Team Plans for PR4:

- HUD v1 refined and tested with live telemetry data from DJI matrice 100
- Dynamically updated obstacle map, tested on flight data sets, running on Jetson TK1
- Mapping node integrated into the onboard computer software
- Conduct ground tests with the aircraft mounted on cart to collect data from Velodyne and DJI flight controller in an environment with dummy obstacles.
- Conversion of 2d obstacle map to fixed size bird's eye view image.
- Audio warnings running on android device/emulator.
- Simple voice commands running on android device/emulator.

My tasks:

- Order wifi adapter for Jetson TK1 and get it to work.
- Publish tf tree and Odom for the mapping node
- Process the telemetry received from Flight controller and publish it for the AR interface node.
- Write the interface for flight data sets and test flight envelope computation and cropbox filter with NEA flight data sets.
- Conduct ground tests with the aircraft mounted on cart to collect data from Velodyne and DJI flight controller in an environment with dummy obstacles.
- Run and test Hari's mapping node on the Jetson with data collected.
- Generate a fixed-size bird's eye view image from the 2d obstacle map.