FlySense

Shivang Baveja

Team C: FlySense

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

ILR 05

November 22, 2017

Individual Progress

Since the last progress review, I have made considerable progress in onboard software development, system integration, and testing aspects. The progress in each of these areas is now described in detail:

1. Onboard software:
   Figure 1 shows the current status of onboard software considering Fall validation Experiment as the goal.

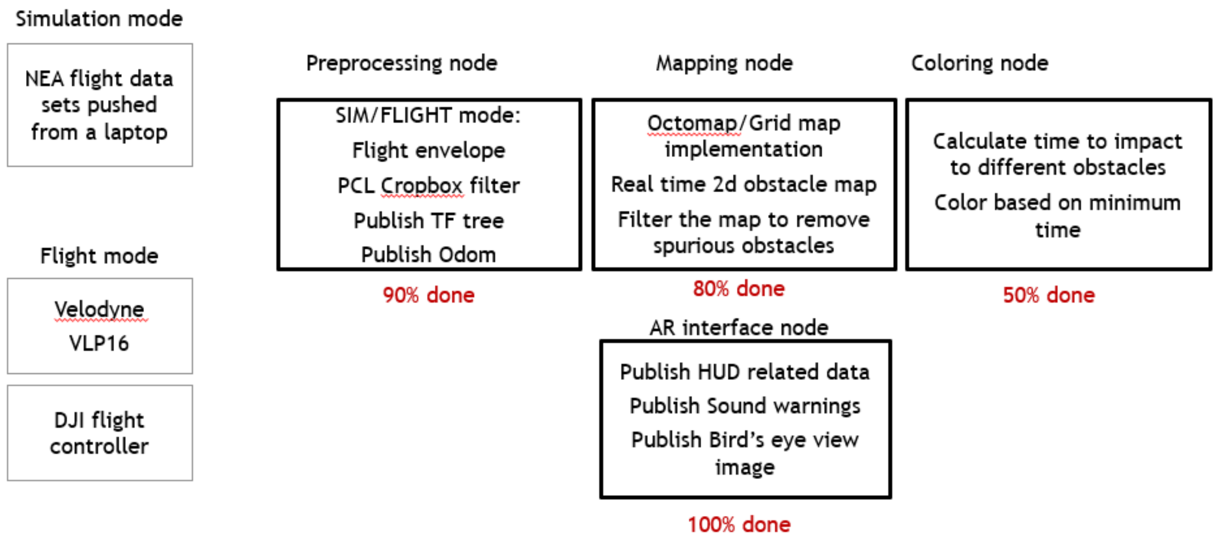

Figure 1 Onboard Software Status

Progress in each of the nodes of the onboard software is described below:

a. Preprocessing node: Three main tasks were completed in preprocessing node.

➔ Publish tf tree for the mapping node: The local position estimator in DJI flight controller must be initialized to start receiving local position estimate. Also, the local position estimate is good if GPS_HEALTH parameter received from DJI flight controller is greater than 3. As soon as GPS_HEALTH value becomes greater than 4, local position estimator is initialized, and that place becomes origin in the world frame.
The attitude quaternion and the local position was used to publish tf transform between world and vehicle and vehicle and Velodyne.

➔ Process the telemetry received from Flight controller, publish it to the AR interface node. The attitude quaternion received from DJI Matrice 100 was converted to roll, pitch and heading angles. This was required for the heads-up display widget in AR headset. In ROS the convention followed for world frame is ENU i.e. (East-North-Up) which is different from NED (North-East-Down) which is followed in Aerospace sector. So, the heading angle was converted to NED.

➔ Dynamic window integration. The use of Dynamic window is to crop the point cloud to keep only the relevant points based on aircraft's flight envelope. This required to reduce the computational load on the onboard computer. Joao wrote the code for dynamic window considering translational dynamics. The model takes in quadcopter flight parameters like min and max speed, max acceleration, etc. Since we are doing ground test only and not flying the quadcopter for the FVE, these parameters were adjusted to match a person moving the cart.

This code was integrated and tested with the data collected on push cart test setup (described in section 3).

Figure 2 shows Velodyne point cloud aligned in world frame from a test conducted outside NSH.

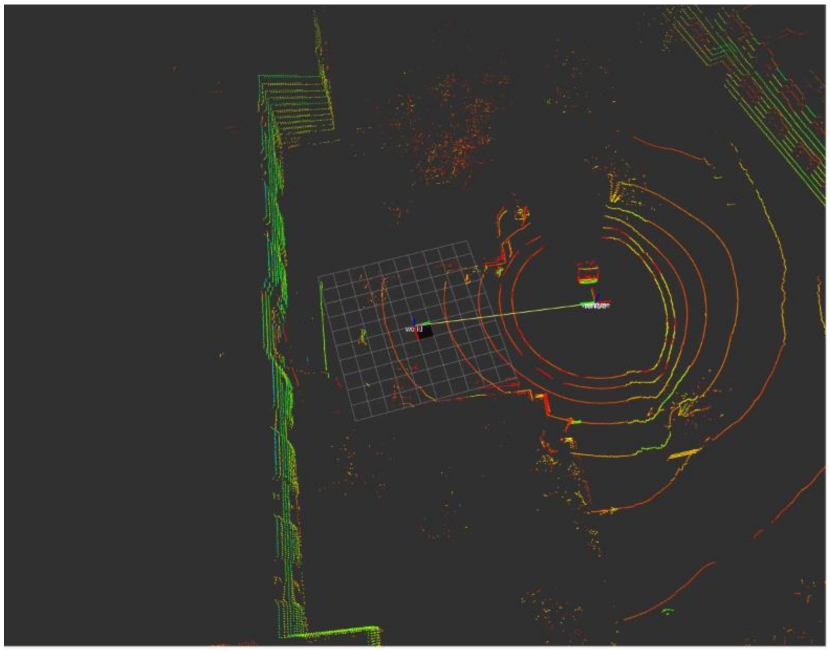

Figure 2 Aligned Point cloud in world frame

Figure 3 shows the cropped point cloud based on the dynamic window. It can be seen that only the relevant obstacle is seen in the point cloud as the push-cart was moved towards that obstacle only.

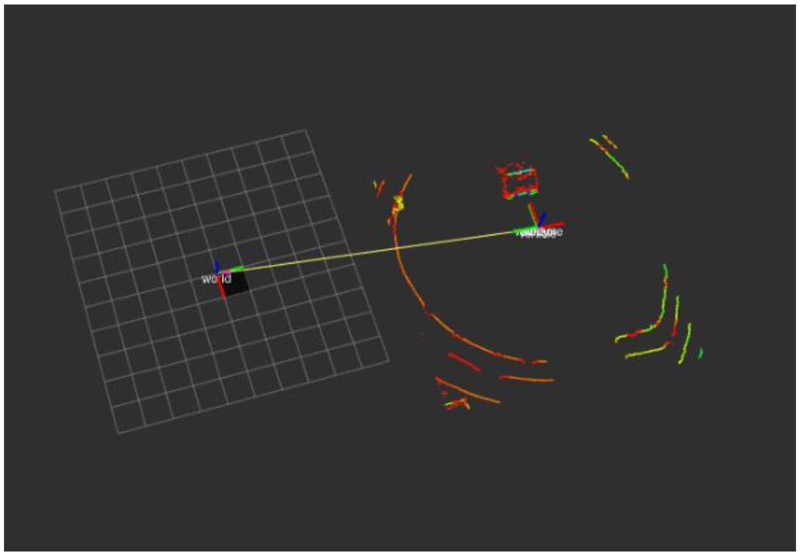

Figure 3 Cropped point cloud based on Dynamic window

- ➔ Another crop box filter was added to filter out the fixed points on the quadcopter frame coming in the field of view of Velodyne. The person moving the cart also comes in the field of view which makes filtering necessary.

b. Mapping node: Mapping node was being developed separately by Hari. We started integration last week and faced some issues. It was seen that when we build the code with C++11 support the PCL libraries crash during run-time. The c++11 support is required for the DJI interface and thus couldn't be removed. We tried building PCL libraries with c++11 support but even that didn't resolve the issue. We have decided to keep mapping node into a separate package for now.

c. Coloring node: Coloring node was developed separately by Nick. We have integrated it into our onboard ROS package.

d. AR interface: The AR interface was modified to include some additional diagnostic information which will be helpful during the spring semester while flying the quadcopter. The new packets included were gps_health and battery percentage remaining.

2. System integration:

a. Jetson TK1:
Last week, we faced couple issues while deploying our latest onboard software on Jetson TK1. The first one was unreliable wifi connection. We had replaced the wifi module that we were using earlier with a new wifi module Intel 7260 mini-PCI card. After following the steps given on NVIDIA forums, I was able to get the wifi module to work. Initial bandwidth testing showed good results. Data transfer rate was seen to be as much as 10MBps, which is good enough for our purpose. After some more testing, I found that wifi module is erratic. Sometimes, it doesn't show up in network

interfaces after boot-up. I tried fixing the issue to no avail. For FVE we have made a workaround by using the wireless router on the moving cart. But this is something that will have to be fixed reliably for our quadcopter flight tests in the Spring semester.

The other issue was DJI interface to Jetson. The serial port connection to DJI Matrice 100 wasn't working initially. After doing some research I had found that USB serial port converter drivers must be enabled in the kernel as they are disabled by default. This means compiling the USB driver support module of the kernel and updating it. After some modifying the driver modules of kernel a bunch of times, the serial port device /dev/ttyUSB0 showed up and was found to be working.

b. FVE test setup:

For our FVE, we will be moving the quadcopter system on a push cart and show the

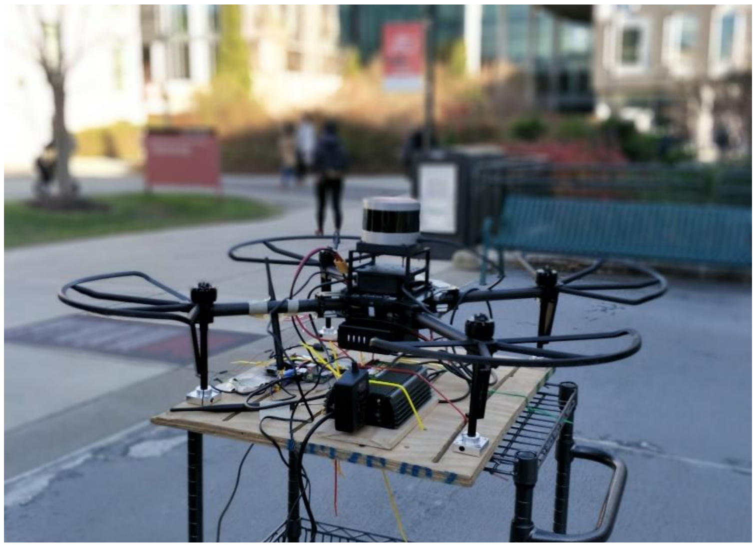
Figure 4 FVE test setup

output on the AR headset. We started building our push-cart after the last progress review. Nick took lead in procuring and assembling all the sub-systems required for pushcart test platform. The power distribution board which is required to power the Jetson and the Velodyne LIDAR was found to be not working. This was holding up our field test with the Platform, so a workaround was devised to get other sub-systems tested. Power distribution board is being debugged separately and will be integrated once fixed.

3. Testing

Aim: To perform a simulation of our Fall Validation Experiment and collect data which could be used for testing our sub-systems.

Setup: The pushcart test platform shown in figure 2 was used for the test and a laptop connected to the systems over WiFi.

Location: Outside Newell Simon Hall (Our decided FVE Location)
Procedure:

- ➔ Move the push-cart near the obstacles.
- ➔ Wait for a few seconds and move the push-cart away.
- ➔ Repeat the process with multiple obstacles and from both left and right sides.

Figure 3 shows the snapshot of how the test was conducted.

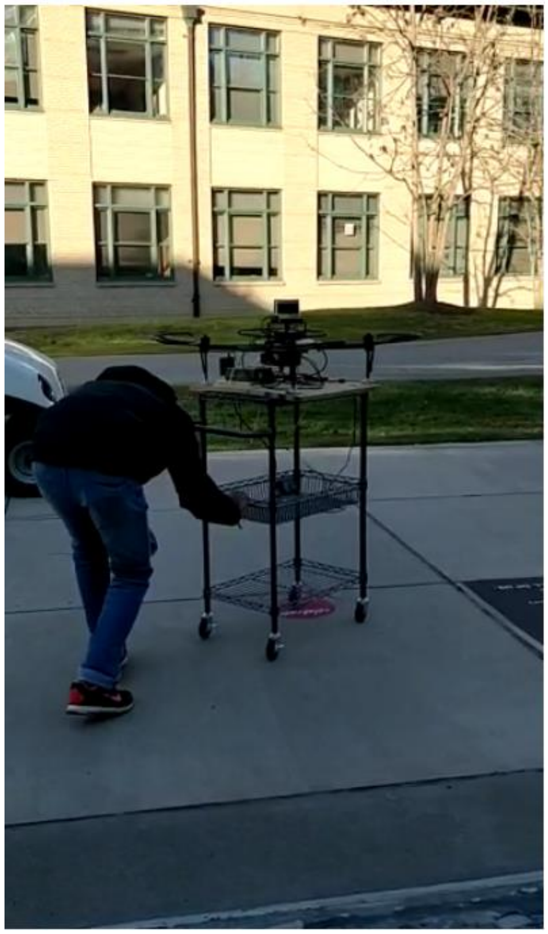
Figure 5 FVE simulation outside NSH

Results:
The point cloud data was visualized in RVIZ. The data was found to be a little jumpy with respect to world frame. With some investigation, it was found that the ROS Velodyne driver is not following the axis conventions defined in Velodyne User Manual. This was corrected in the code and the results were much better.
Figure 4 shows the point cloud with decay setting in RVIZ set to 1 second. This tells us that the point cloud data is getting properly aligned with the world frame.

Even after aligning the point cloud in world frame, there is some drift. This is due to position and orientation inaccuracy. To correct the alignment further, we can add registration step in our onboard software pipeline. Hari is conducting some tests to check if would need registration step or not. The alignment error can be seen in Figure 5, which shows the same point cloud with decay equal to 5.

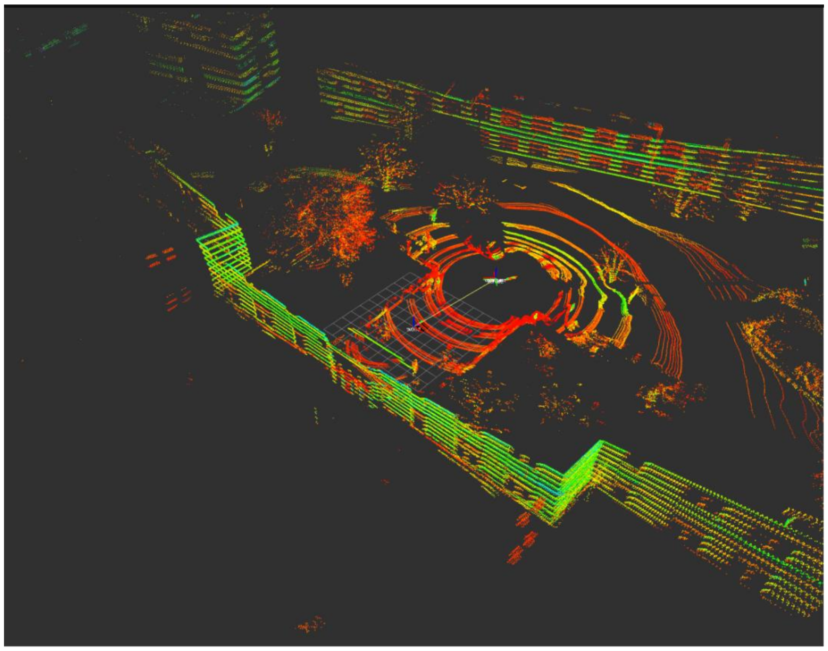

Figure 6 Point cloud aligned in world frame, decay 1 (Outside NSH)

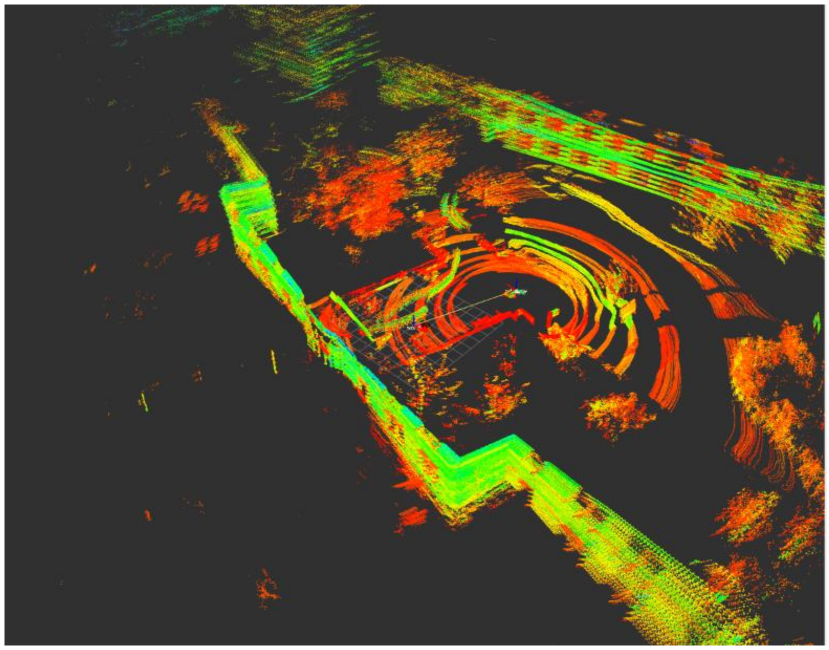

Figure 7 Point cloud alignment error, decay 5 (Outside NSH)

Challenges faced

- While integrating the mapping node into the Onboard software, we faced some incompatibility issues. The DJI SDK needed C++11 compiler option enabled whereas the

PCL nodes were crashing when compiled with that option. We tried resolving it but ultimately decided to keep the two codes separate for now.

- The serial to USB cable which is used to interface to DJI flight controller was not getting recognized on Jetson TK1. It took me some time to figure out there's no driver installed, and I'll have to build driver as a Kernel module and update. Luckily it all worked out.
- Communication from Jetson (ROS) to AR (ROS Java) over the network was giving issues. This was primarily due to limited configuration capability in Epson BT-300 AR headset. We were able to get it to work after trying multiple network configurations and settings.
- As usual, testing in the cold. We now have one set of recorded data of our FVE simulation which means we don't need to go out in the cold as much.

Teamwork

Name	Contribution
Nihar Tadichetty	• Updating the ROS interface and configuration to communicate reliably with Jetson over network • Fix issues with Image reception • Pocket Sphinx voice command library integration and testing • Audio warnings integration.
Joao Fonseca Reis	• Code for dynamic window based on flight envelope • Voice commands using Google API and reliability testing • Android code to generate sound warnings • Complete the algorithm to find the nearest obstacle based on flight envelope and determining if the object is on the left or right.
Harikrishnan Suresh	• Completing the first version of mapping • Implemented Statistical outlier filter, Voxel grid filter, a height map and octomap. • Testing and tuning mapping parameters with NEA flight data sets • Testing and tuning mapping parameters for our FVE test platform. • Helping Nick with coloring node • Helping me with FVE simulation test
Nicholas Crispie	• PCB parts procurement, assembly, and testing

	• Setting up the Pushcart test platform • 3d printed part for Lidar mount • Completed first version of coloring node which subscribes to 2d map, converts it to a bird's eye view image and publishes it

Plans

Team Plans for FVE:

- Implement sound warning generation code in onboard software and integrate it with AR controller.
- Complete Bird's eye view image generation.
- Tune the mapping parameters based on flight data from NEA and our FVE simulation.
- Conduct more FVE simulations and pre-validate our sub-systems and complete system.

My tasks:

- Test onboard software interface to AR controller and make it reliable for FVE
- Implement Sound warning algorithm developed by Joao
- Help Nick in completing the coloring node
- Help Hari in completing the Mapping software
- Conduct more FVE simulations and pre-validate our sub-systems and complete system.