Angad Sidhu

Team B Arcus
Clare Cui
Maitreya Naik
Logan Wan
ILR 4
11/11/2016

Progress

These last two weeks I have been spending my time working on software and getting things ready for the data collection procedure. We all went out to LaFarge quarry to collect more data. This time we collected about 5 bags of data of three loops in the quarry with LiDAR, GPS, IMU, and RGB data. In order to prepare I worked on preparing the calibration software between the LiDAR and Camera. This software was surprisingly difficult to get running because of it's reliance on OpenCV 3.1 which provides support for fisheye lens. This involved a lot of understanding of the underlying build system of ROS to make sure it included OpenCV 3.1 instead of 2.4 and recompiling an opencv dependent ROS libraries with OpenCV 3.1, instead of 2.4, called cv_bridge. Below is data from a collected calibration bag visualized with ros' image_view and rviz. Figure 1 shows an rgb image of a calibration board of april tags and figure 2 shows the lidar point cloud taken of the exact same scene. The software for LiDAR-Camera calibration will work by extracting the board plane in the rgb image and the board plane in the lidar cloud and construct a transform between the sensors that explains the difference between these two planes.

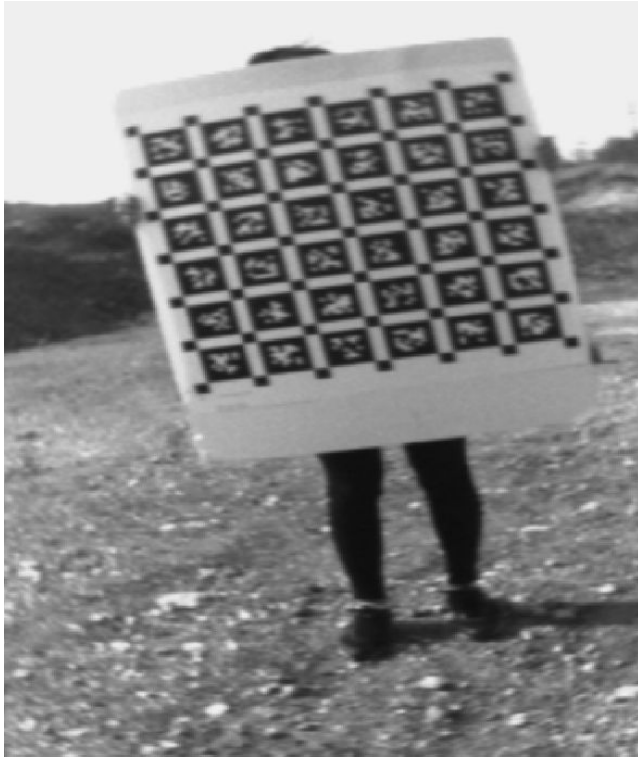

Figure 1

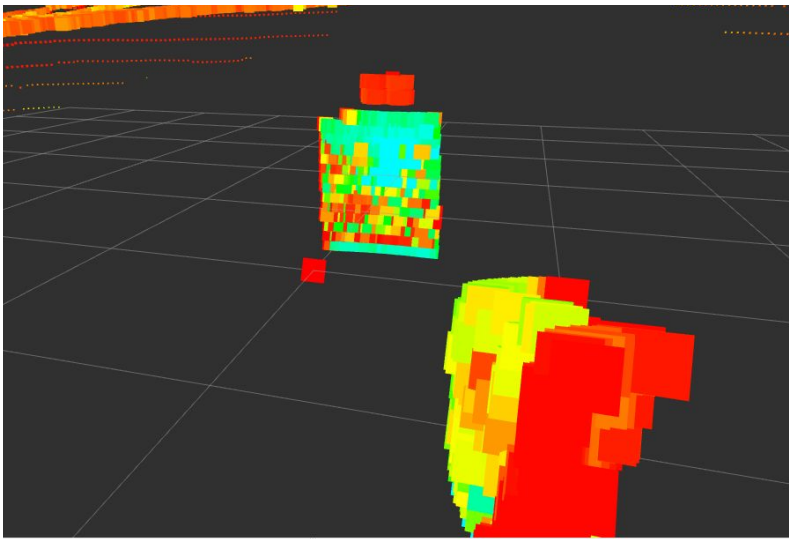

Figure 2

These two weeks I spent a lot of time aiding with calibration, trying to help Maitreya get the software up and running on his computer, testing out Kalibr with him and trying to determine an appropriate calibration procedure to reduce the time it takes to calibrate. Included in this, I also devised a few different launch scripts for calibration which should reduce the time it takes

to set up for calibration. I also made modular launch scripts for device acquisition so that these can be plug and played into other launch scripts to save time in writing further launch scripts. Essentially I wrote launch scripts for imu, gps, lidar, and camera devices. They each, individually, bring up the software drivers for each of their respective devices. I then reference these launch scripts, with configurable parameters for settings things like exposure time, data capture frequency, bit format, port, etc, in other larger launch scripts that perform a functional task like calibrate.

I also spent some time trying work on my odometry performance script but decided early in this cycle to pivot as it was not a major task and the work I was doing for it was not applicable to the new bags of data we were going to collect. The issue with the old bags is that they lacked any sort of global orientation. With IMU we have a pretty good global estimate of orientation which should make it much easier to align the odometry data we have now collected with the GPS data.

The last bit of work I spent this week was attempting to restructure our code base so it can be more easily integrated into our advisors VCS solution. This was an unexpected requirement that we encountered but was fairly simple to accommodate. Essentially all I had to do was convert packages of code into their own repositories and add a central repo. The central repo contained only a configuration file and a update script which pulls down repos specified in the config file and builds them.

Challenges

The biggest challenge I faced this week was trying to convince the ROS build system to link OpenCV 3.1 instead of OpenCV 2.4. This is extremely complicated as there are numerous layers to the build system which obfuscate what is happening at the low level. The first step was modifying the CMakeList in my package and setting it to search for the exact version, 3.1, of OpenCV. However, strangely, this seemed to have no effect when using catkin_make. After a bit of time searching I discovered that the issue could very well be due to CMake caching the OpenCV package information it had found earlier during the compilation of other packages. By removing these entries from the cache I was able to get cakin_make to build my package with OpenCV 3.1. Another solution I believe would work is just using catkin tools' "catkin build" command. This command executes the building procedure of each package in its own cmake process i.e. effectively sandboxing each package. This would prevent CMake from caching any found libraries between compilation of different packages.

Teamwork

Clare spent time working on designing and CADing the sensor mount for our UAV. This was a critical task in order for us to complete hardware integration on time. She and Logan also worked on a UAV mount that we took with us on our data collection to keep the UAV at least 10 feet off the ground while collecting data. Logan worked on project management tasks like

getting all the tasks migrated into Asana. He also worked on printing out the sensor mount and aided in CAD to get it slightly more amenable to 3d printing. Maitreya was heavily focused on camera and imu calibration. He investigated a number of different packages to get calibration running.

Plans

For this next week I plan on getting the repository into a working state and catching everyone up with all the packages they need in order to begin development on their various software tasks. I also plan on working on BLAM loop closures, but the priority of that task is not as high as some other setup tasks that I will be working on. I hope to get our onboard platform setup and loaded with the latest software and making sure all the drivers are running. Also plan on doing specific setup tasks like getting ssh server running and setting up udev rules for the different devices so that we can count on devices having predictable addresses. Also will be trying to get networked setup running across different machines so we can visualize information on a base station which will be essential for our FVE.