Individual Lab Report 11

By Clare Cui

Team B: Arcus

Clare Cui

Maitreya Naik

Angad Sidhu

Logan Wan

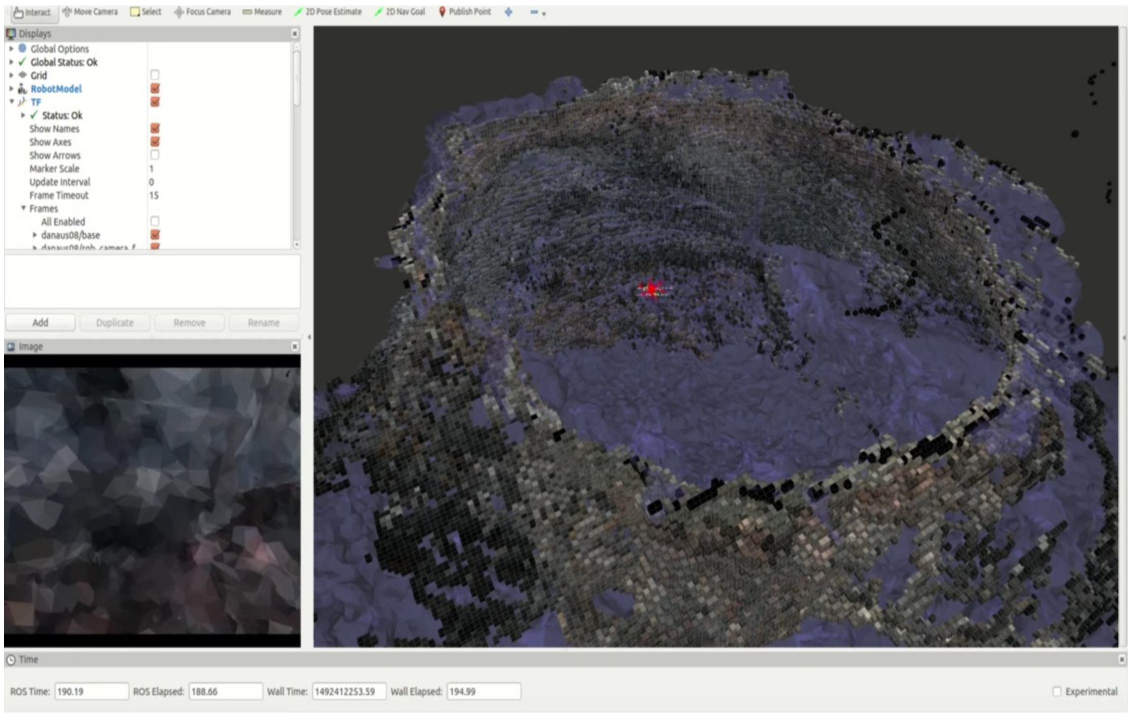
18 April 2017

Individual Progress

Last ILR, I had mentioned that I was hoping to get mapping and visualization with bagged data and working on obstacles. After discussing with the team a little more, Logan decided to take on the obstacle construction because he had finished his efforts with observation synchronizer and was ready to work on a new aspect of mapping. Ricky took on integrating real data with our sandbox as there was actually a lot more involved in getting the data integrated that I had originally thought. This change was for the best because, while I was able to colorize voxels, there were still some issues with mapping to work through and troubleshoot.

After initially getting the colored occupancy grid map visualized, it was noted that a lot of white voxels were showing up erroneously and, additionally, the colorized grid was not aligning with the contours of the actual map. Logan had also hard-coded fake values for the camera intrinsics matrix and transformation matrix. To begin, I used a parameter_utils package to extract the camera intrinsics as well as the transformations for the camera and Velodyne to the body frame from yaml files in the map_server package. I found a bug where we were transforming from the Velodyne frame to the RGB frame but never transforming back to the Velodyne frame, which was what the rest of the mapping pipeline expected in order to transform the points into the world frame.

After finding these issues, there was also the issue where some random white voxels would appear in the colorized set. Angad helped me realize that there were some issues in map_utils, where I had forgotten to set the type of a variable to be a float. Originally, it referenced a type that was “typedef'd” to a float, which I modified to typedef to a struct instead. Interestingly enough, this bug did not have any issues compiling, which was part of the reason it was difficult to catch. After resolving this, we discovered that the Velodyne point cloud was not transforming correctly to the camera frame. This was probably the toughest item to debug since we were convinced that our math was right. This is more fully documented in the next section, Challenges. After this, however, we were finally able to visualize the colorized occupancy voxel map. The final map can be seen in Figure 1 on the next page.



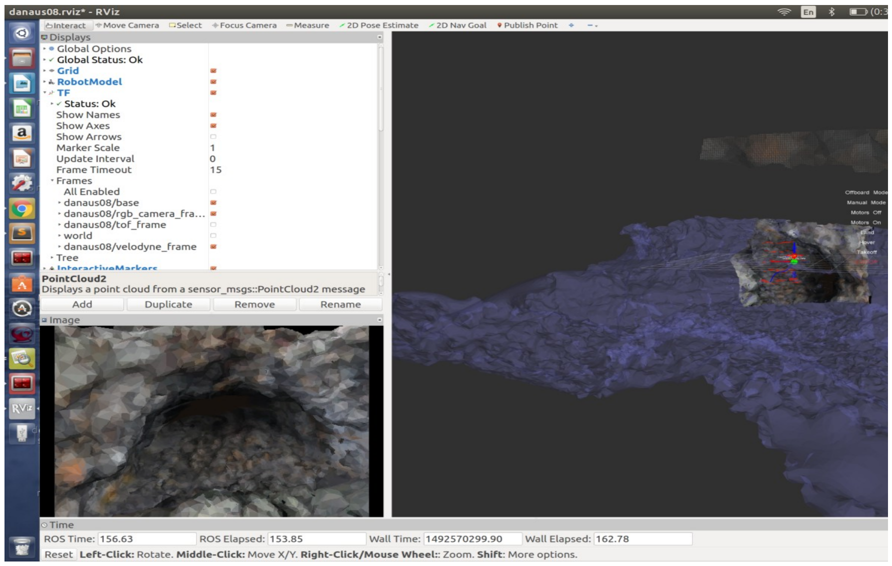
```
Global Options
✓ Global Status: Ok
Grid              ☐
RobotModel        ☑
TF                ☑
  ✓ Status: Ok
  Show Names      ☑
  Show Axes       ☑
  Show Arrows     ☐
  Marker Scale    1
  Update Interval 0
  Frame Timeout   15
  Frames
    All Enabled   ☐
    danaus08/base ☑
```

```
ROS Time: 190.19   ROS Elapsed: 188.66   Wall Time: 1492412253.59   Wall Elapsed: 194.99   ☐ Experimental
```

Figure 1*: Current state of the visualization pipeline. RGB voxels are accurately being displayed in the occupancy grid map.*

Challenges

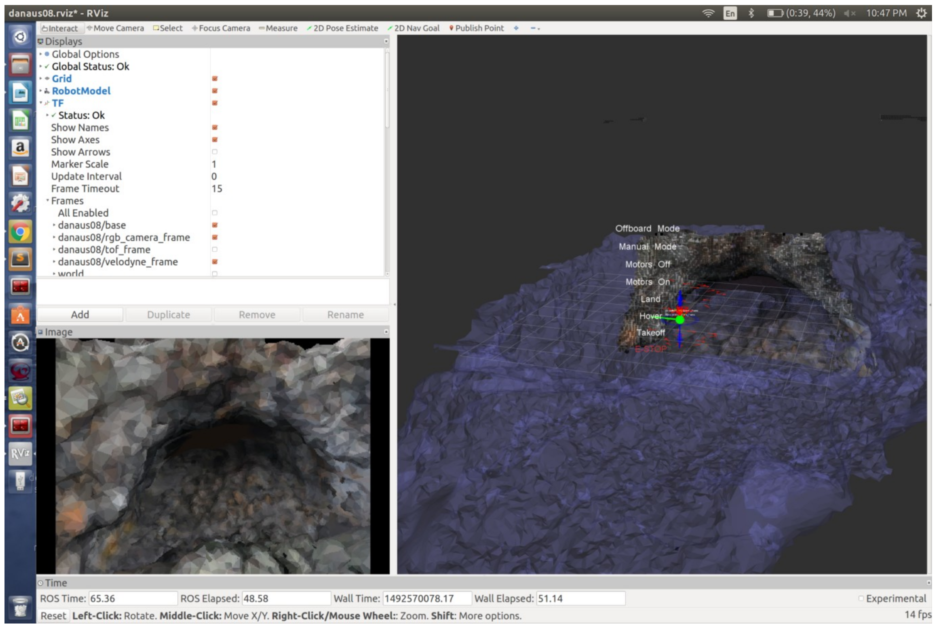
A major challenge that I faced this week was the last few steps of getting the Velodyne points properly transformed into the RGB camera frame, and I worked with Angad to debug it. For awhile, we played around with the transformations as well as the limits on the image frame. I was taking the inverse of the Velodyne-body transformation and right-multiplying the camera-body transformation to get a Velodyne point to the body frame and then to the camera frame. I was also playing with the limits on the image plane, as it appeared that the image plane was not properly centered or it was clipped for certain limits. To get a better visualization, I made the Velodyne simulator projected point cloud much more dense, so that I could see the exact pixel values that were being seen by the camera and transformed from the Velodyne frame. An example of this is seen in Figure 2 below. This definitely helped us make some more educated guesses to troubleshoot the bug as we were able to exactly correlate the image with the map itself, seeing how the points needed to be rotated or flipped and approximately how much they were being displaced by.



```
Global Options
Global Status: Ok
Grid
RobotModel
TF
  Status: Ok
  Show Names
  Show Axes
  Show Arrows
  Marker Scale      1
  Update Interval   0
  Frame Timeout     15
  Frames
    All Enabled
    danaus08/base
    danaus08/rgb_camera_fra...
    danaus08/tof_frame
    world
    danaus08/velodyne_frame
  Tree
InteractiveMarkers
PointCloud2
Displays a point cloud from a sensor_msgs::PointCloud2 message
ROS Time: 156.63   ROS Elapsed: 153.85   Wall Time: 1492570299.90   Wall Elapsed: 162.78
```

Figure 2: *An incorrectly transformed Velodyne point cloud. The point cloud is flipped across the lateral dimension and is rotated 90 degrees clockwise. Making the point cloud denser helped us to see these differences clearly.*

Eventually, Angad discovered that the transformation matrices were not being multiplied correctly by looking at Wennie's (our PhD advisor) code for a similar transformation. The transformation should have a left multiplication of the inverse camera-body matrix to the velodyne-body matrix. Addiionally, we did have to adjust the limits of the image frame, so some of the assumptions Logan had made earlier were not correct. The rectified point cloud can be seen in Figure 3 overlaying the cave that is depicted in the image sent through the RGB camera topic.



```
Global Options
Global Status: Ok
Grid
RobotModel
TF
  Status: Ok
  Show Names
  Show Axes
  Show Arrows
  Marker Scale      1
  Update Interval   0
  Frame Timeout     15
  Frames
    All Enabled
    danaus08/base
    danaus08/rgb_camera_frame
    danaus08/tof_frame
    danaus08/velodyne_frame
    world
ROS Time: 65.36   ROS Elapsed: 48.58   Wall Time: 1492570078.17   Wall Elapsed: 51.14
```

Figure 3: *Corrected point cloud transform with corrected sensor limits. This matches the RGB camera output.*

<u>Teamwork</u>

Logan Wan: Logan worked on the obstacle simulations – columns that we can import into our mapping for the navigation pipeline – and worked on getting the UAV running with the sandbox.

Angad Sidhu: Angad helped me debug some of the mapping errors and worked on integrating bagged data with the sandbox using OpenCV. He also integrated the SLAM pipeline into sandbox and wrote a test script to visualize mapping with the ground truth compared to SLAM odometry.

Maitreya Naik: Maitreya worked on the trajectory planner and wrote the RRT planner using utilities from the RASL code base, tested it in the sandbox, and covered most edge cases. He also wrote test script for a dummy example for reviewing by our advisor.

<u>Future Plans</u>

For SVE, I plan on resolving any last issues with the mapping. There are some extraneous black voxels that do not belong in the map, so those need to be removed, possibly by modifying and implementing a preexisting Velodyne trimming function. Additionally, I need to publish a point cloud that only shows occupied voxels that are also colored. Currently, the pipeline does not distinguish between the two, and, once the cell is shown as occupied, it turns white, which ruins the corrupts the true visualization.