INDIVIDUAL LAB REPORT 7

Progress Review - 8

Shubham Garg

TEAM H (PHOENIX)

Teammates:

Akshit, Parv, Steve

September 26, 2019

Individual Contribution:

1. Mounting and testing multiple Realsense cameras on the Husky
2. Designing the CAD models of all the mounts for adding it in the Husky's URDF
3. Stitching point clouds from different stereo cameras

Mounting and testing multiple Realsense cameras on the Husky:

RealSense T265 is a tracking camera that is designed to be more optimal for Visual Odometry and SLAM (wider field of view and not using infrared light). It can do SLAM on board as well as loop closure. However, this camera is not able to return RGB images (since it does not have an RGB camera onboard) and the depth returned is not as good as the D435i camera. So, using both a RealSense D435i sensor and a RealSense T265 sensor can provide both the maps and the better quality visual odometry for developing a full SLAM system. The D435i used for the mapping, and the T265 for the tracking. So, we have one such pair of the camera attached exactly at the center of the husky.
We have additionally used two D435i on the sides of the husky for nearly 180 field of view. I designed the mounts for the camera and attached to the base plate of the husky (I also got help from Kevin, a student from Prof. Oliver's Lab). As shown in figure 1, we have placed the camera in front of husky such that we can achieve the maximum field of the view from four cameras.
Next major task was to test if the Intel NUC can process/handle the streams from all the four cameras. To validate that, I wrote a C++ script using librealsense SDK APIs to stream the data from all the four cameras based on their serial numbers. Based on the results, we found that Intel NUC can stream all four depth images along with its point cloud with around 60% CPU usage.

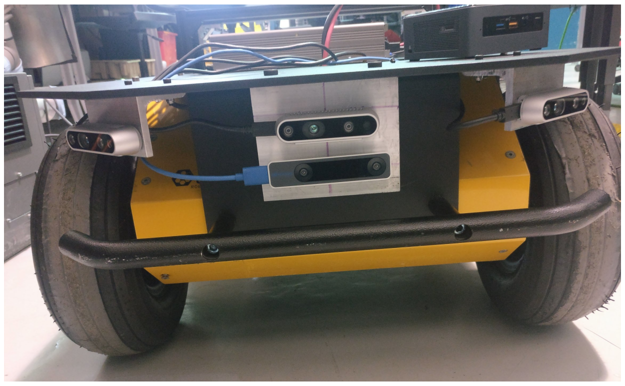

Figure 1: Front view of the Husky with the mounted cameras

If we are using librealsense SDK APIs, then we can't use a ROS interface to subscribe the depth images/point clouds data from the camera as the device gets busy. So, I wrote a python script to subscribe the depth images/point cloud data from all the cameras so that multiple nodes can subscribe to difference ROS topics from the camera simultaneously.

Adding four Realsense camera with the robot (Husky) URDF:

Initially, we tried calibrating the multiple cameras simultaneously using Kalibr tool but the result was not satisfactory because the target (April tag) was not flat and we were moving target instead of the Robot (husky) to calibrate the cameras. So, we used husky customization ROS package to compute the camera transforms to the base-link by importing the STL files in the URDF settings of the robot model. Hence, we need not worry about any of the transforms between any link on the robot and the camera. Also, we are planning to mount UR5 arm on the husky, having TF tree will make the transform between the camera and UR5 easier.
Adding the camera joints/links in the existing robot model was challenging as I had to manually tweak the translation and rotation parameters to exactly align the camera mounts to their exact location on the robot base plate as shown in figure 2.

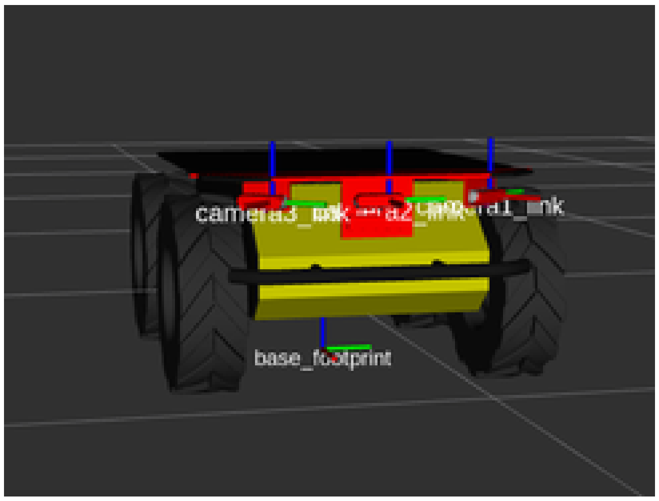

Figure 2: Husky URDF with four realsense cameras

Stitching point clouds from different Realsense camera:

In last semester, we used SICK LiDAR for creating the occupancy grid map which is used by the robot localization ROS package for obstacle avoidance and creating the map of the environment. But since we are now planning to mount UR5 arm also on the husky, we couldn't find a suitable location for LiDAR on the husky as it will obstruct the UR5 workspace.

Now, to create the 2D occupancy grid map we are stitching the point cloud from all the three D435i cameras. Point cloud data from all the cameras are subscribed and multiplied with the camera base-link transform. The combined point cloud as shown in figure 3 is then converted into a laser-scan so that we can use our existing obstacle detection code and also we can use the Husky gmapping package which generates cost maps for the local planner. The output of the cost map is fed to the move base planner which executes the trajectory.

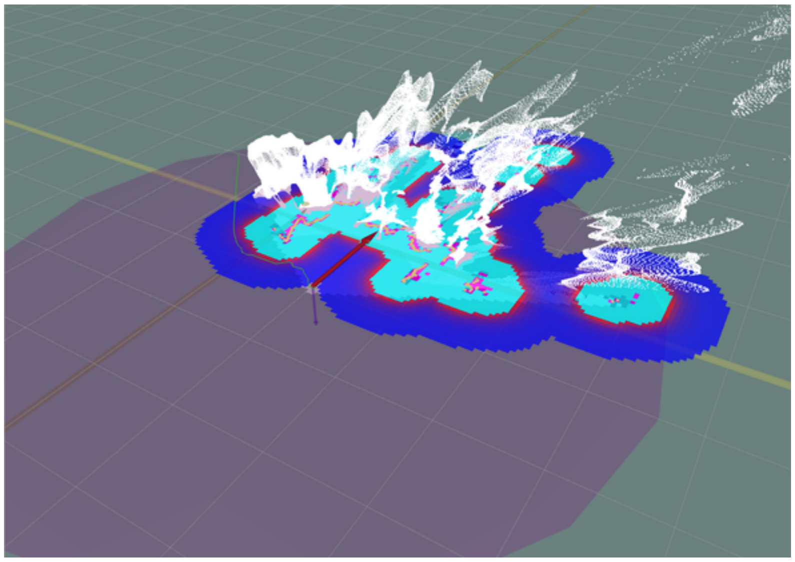
Figure 3: Stitched point cloud along with the cost-map

Challenges:

Challenges faced in the last two weeks are discussed below:
1. Adding the camera mounts to the URDF and TF trees was challenging as we had to figure out the exact translation and rotation parameters for all the camera links/joints.
2. We faced multiple difficulties in visualizing the point cloud because we were not publishing the correct parent and child frame ids of the combined point cloud.
3. We faced multiple testing issues as the Realsense output was getting stopped when we were trying to collect the ROS bags.
4. We tried different multiple camera calibration packages but they didn't work as per expectation and we had to setup the husky customization ROS package to create a TF tree.
5. DJI default disparity images are not usable and we had to manually fine-tune parameters for semi-global matching.
6. We spent a lot of time in fixing the compilation errors while setting up the AirLab stack.

Teamwork

Initially, we all brainstormed the software pipeline for the husky to enter the door. And we decided to use point clouds generated by the Realsense for obstacle avoidance and T265 odometry output for robot localization.
Shubham worked on deciding the location of cameras and mounting it on the husky base plate. **Parv** helped me in debugging the issues faced in stitching the point clouds. **Akshit** suggested me beforehand that the ROS and the librealsense APIs can't work together which made me write all the code with the ROS interface (found really helpful later).
Parv and **Akshit** worked on testing the image based visual servoing on the simulation.
Akshit was responsible for setting the Manifold 2C for DJI drone with the Airlab's core-autonmy stack. He also tested the opening detection algorithm on the DJI such that it can align itself exactly in front of the opening and testing is still ongoing.
Parv was responsible for setting up the gmapping and robot localization ROS node such that it can subscribe the 2D occupancy grid map generated by the combined point cloud.
Akshit and **Shubham** helped him in testing the complete software pipeline on the husky.
Steve worked on the door detection using depth images but he faced issues with the algorithm as door doesn't have boundary at the bottom and we are just not able to distinguish between a window and a door.

Future plans

1. **Shubham** and **Parv** will be working on improving the frame rate of the combined point cloud data.
2. **Shubham** and **Parv** will be working on the sensor fusion (fusing odometry data from the tracking camera T265 and 3DM GX5-45 IMU) using Robot localization ROS package.
3. **Akshit** will be responsible for testing image based visual servoing on the DJI drone for it to enter through the window.
4. **Steve** will be responsible for door detection algorithm using depth images/point cloud data.
5. **Shubham** will be responsible for selecting the proper WiFi router and working on testing the range of the same.
6. **Akshit** will be working on setting up the communication between the robots using WiFi
7. **Steve** will be responsible for setting up the database for the information that will be shared between robots.