Nobleo Technology,
Eindhoven University of Technology

Internship Thesis - The Follow-up

Author:
Josja Geijsberts

June 21, 2019

1 Implemented changes

Based on the discussion of the results in the internship thesis, various improvements to the existing implementation are suggested. In the period after finalizing the internship, three improvement have been integrated.

1.1 The `dory_simulation` repository

The changes discussed in this section are applied in the `add/transform_pointcloud` branch.

1.1.1 Switch to different laser data type

The use of the ROS `LaserScan` message type was a convenient data type for processing the LiDAR output. However, for different LiDAR's and for a better representation of the environments this data type is not ideal. Using the ROS message type `PointCloud` allows for a 3D representation of the data and is more robust for using different LiDARs with different ranges. Most LiDAR output uses the `LaserScan` data format, so a conversion has to made from the `LaserScan` format to the `PointCloud` format. This conversion is done through the `transformLaserScanToPointCloud()` function of the `LaserProjection` class from the `laser_geometry` ROS package. This conversion is currently performed in the `tf_prefix_stripper.cpp` file of the `dory_simulation` repository, which definitely is not the right place.

1.1.2 Filter laser data

During the internship, it was found that most LiDAR data is very noisy, which leads to incorrect detected obstacles. Therefore, a form of filtering was required. This filtering is commonly performed by a filter provided by the `pcl` ROS package. To apply this filtering, the ROS message has to be converted from the `sensor_msgs::PointCloud` format to the `pcl::PointCloud` format, which can be done using the `pcl_conversions` ROS package. After the filtering is done, the data is converted back to the `sensor_msgs::PointCloud` format. This implementation perhaps seems messy and inefficient, using a commonly integrated filter, however, also has its advantages. Since the method is commonly used and applied, the filtering method and implementation have been thoroughly tested which implies high performance. Therefore, a PCL filter is implemented that filters out the statistical outliers in the LiDAR output. This part is again integrated in the `tf_prefix_stripper.cpp` file of the `dory_simulation` repository, which is not the correct place for these forms of data processing.

1.2 The `dynamic_obstacle_avoidance_layers` repository

The changes discussed in this section are applied in the `cloud_clustering` branch.

1.2.1 Clustering of laser data

The method used to cluster the LiDAR data to describe separate objects that was developed during the internship was not compatible with the `PointCloud` data format, due to the fact that the data is not radially sorted. Therefore, a new implementation for data clustering is integrated. Again, the `pcl` ROS package is used, more specifically, the `EuclideanClusterExtraction` class. This new form of clustering is implemented instead of the original approach and can thus be found in the `boats_from_lidar.cpp`.

1.3 Logged obstacle removal

As described in the internship thesis, static obstacles are logged and thereby kept in the obstacle map for a user-specified period of time. When dory (or any robot using the `dynamic_obstacle_avoidance` layers) gets close to a logged obstacle, the sensor data is used to check whether the logged obstacle is still there. If not, it should be removed from the logged obstacle list and thereby from the obstacle map. This process relies on the `LaserScan` datatype, so by converting the datatype from

`LaserScan` to `PointCloud`, the process of checking logged obstacles no longer works. There are various ways to tackle this issue:

1. Loop through all `PointCloud` clusters and compare their centre point with the logged obstacle, if the centre point is sufficiently close, keep the logged obstacle and update its properties.
2. Loop through all `PointCloud` points and check if there is a point sufficiently close –> not very representative
3. Loop through all `PointCloud` points and check if sufficiently many points lie within the logged obstacle –> still quite inefficient, is it even possible?

All of these solutions have downsides; for solution 1. this means that within the static layer, clustering also has to be performed which is not very efficient. This is, however, considered to be more efficient than looping through all point and it is undoubtedly more scalable. Therefore this method has been implemented. The implementation has sadly not been finished, as the result is not yet performing as it should.

2 Suggested improvements

Whilst implementing various changes mentioned in the internship thesis, a few additional required changes have been discovered. The newly discovered required improvements as well as other next steps are discussed here.

2.1 Move conversion changes

As mentioned in the previous chapter, various steps have been taken to allow for the use of the `PointCloud` datatype instead of the `LaserScan` datatype. These steps, however, are not implemented at the right location. These should definitely be moved to a better place. The most apparent location would be the `dory` repository and the `dynamic_obstacle_avoidance` repository would be a suitable alternative.

2.2 Logged obstacle removal

The implementation of the logged obstacle removal has sadly not been finished, as the result is not yet performing as it should. The result should show that obstacles near the robot are compared with the data and optionally removed, and obstacles far away should remain logged as only close data should be considered as reliable. This part does not seem to be working correctly yet. Note that this implementation does not take into account that occluded obstacles (so obstacles that the robot can not see since there is something in the way) are not present as clusters in the data, and therefore these will be removed.

2.3 Improve velocity estimation

The estimation of the velocity is reasonably accurate at constant velocities and after sufficient time. This process, however, takes time (so quite some measurements) and does not work for less steadily moving obstacles. This has already been mentioned in the internship thesis, but further testing has proven that this should definitely be improved.

2.4 Thorough testing

Testing has only been performed once with Dory, and various issues were encountered that blocked proper testing. On Clara more testing has been performed, which yielded valuable results. It is therefore of high importance that more tests are performed with Dory.

2.5 Investigate safety risk data-filtering

Filtering the lidar data is necessary for the obstacle detection to work reliably. This filtering removes data points which improves consistent clustering. This, however, also means that potential small obstacles are filtered out, and are not represented on the costmap. This could result in collisions. Therefore, the safety of implementing this filtering should be investigated