Integration process of new Lidar

A collection of actions and changes made to accommodate for a new Lidar

Thomas Schenk

Contents

[Foreword 3](#_Toc136292995)

[Context 4](#_Toc136292996)

[Problem 5](#_Toc136292997)

[Goal 6](#_Toc136292998)

[Information about the lidar 7](#_Toc136292999)

[Basic installation 8](#_Toc136293000)

[Changes made 9](#_Toc136293001)

[Incompatibility with the exploration stack 10](#_Toc136293002)

[Robot colliding with obstacles 13](#_Toc136293003)

[Final validation 16](#_Toc136293004)

## Foreword

This document is a collection of multiple smaller “products” I previously produced. However, after some time it became apparent that these products were all very similar to each other; the problem remained the exact same. Therefore I decided to group all of these into one document. Not only to increase the validity and testing capabilities of my work, but also to give the next group a more efficient carryover in the Lidar department.

## Context

The Jackal uses lidar data to map its surroundings. Together with RTAB-map, it can create a 2d map of the environment around this. The quality of this map is usually dependent on a few factors, most notably: the lidar used and its quality. The project group received a lidar which is far more accurate and powerful than its previous one.

## Problem

The current lidar is old and inferior to the newer lidar, it needs to be changed out. However, a lot of processes currently working perfectly fine are dependent on the lidar. So not only does the new lidar need to be integrated, changing the lidar to a new one means all these processes need to be reconfigured.

## Goal

Integrate the new lidar so that all processes work seamlessly with the new lidar, however dependant these processes may currently be of the old lidar.

## Information about the lidar

The new lidar we received is the VLP-16 velodyne lidar. Like its name suggests, it has 16 lasers which emit from the base of the lidar. The fact that this lidar uses multiple lasers, means that it can not only portray 2d imagery; but also 3d!

This lidar opens up a wide range of new applications to the project, which may not emerge now; but could be enormously beneficial in the future.

A close-up of a device

Description automatically generated with medium confidence

Picture of the new lidar and its controller.

## Basic installation

Like many processes in ROS, this lidar has its own package. To install this, there is an official guide in place which guides the user through this process. I will not indulge into the specifics of this, since everything and more is already covered in the guide.

You can find the guide here: <http://wiki.ros.org/velodyne/Tutorials/Getting%20Started%20with%20the%20Velodyne%20VLP16>

This guide assumes no jackal is present and explains the basics of installing their standalone package and visualizing it quickly in RVIZ.

The jackal also needs this package, it is best placed in a separate catkin workspace. Or in a place where you gather your libraries. Don’t forget to make after you added it.

## Changes made

This chapter will discuss changes made to the software to accommodate for the new lidar after the installation was completed, like mentioned in previous chapter. These changes will be handled much like separate problems, yet with a sub-problem to define their problematics a bit more specifically.

### Incompatibility with the exploration stack

#### Context

One of the applications of the lidar is to provide the exploration stack with useful data about all sorts of things; objects, surroundings and in some cases even positional data. The exploration stack therefore always knows what the lidar sees and interfaces with it almost directly.

#### Sub-problem

The old lidar was of a different make and parameter set than the new lidar, therefore; the exploration stack interprets the current lidar data completely wrong/not at all. To ensure its functionality again, we need to tell the exploration stack that there is a new lidar to listen to; and to interpret its data accordingly.

#### Goal

Implement the new lidar software into the already existing exploration stack so that the mapping and exploring features work like intended again.

#### Launch files

The robot executes all its real-life functions via launch files. These are files which determine what is executed but most importantly in which order these executions are done. For autonomous exploration you will mainly use two of these launch files.

* Exploration.launch

This launch file launches all the software connections between hard- and software. This is the core of the autonomous exploration filesystem, which links everything together. In strict terms, this is the only launch file you need to launch in order for the robot to explore.

* Visualization.launch

While practically it is true that the robot only needs the *Exploration.launch* file, for us developers (or end users) it is nice that we can see what the robot sees. This is where the visualization file comes in. It takes every datapoint the robot publishes and maps these in Rviz; a visualization program for ROS based applications. This tool is nice for when you don’t have a direct line of sight to the robot, or to “dashboard” all the robot outputs in a kind off GUI like environment.

These launch files contain all the necessary information to launch virtually anything, from new ROS nodes to running bash scripts. For the velodyne we need a separate ROS package, that means we also need to launch this new package.

The first step of solving the problem is to locate in which launch files lidar nodes are launched,

#### Changes made

To accommodate for the change in hardware, we only need to edit the *Exploration.launch* file. This is the only launch file which launches anything lidar related. Luckily, that means this is a quick fix.

The only changes really needed are found in the sensor category, we need to swap out the working directory which is used when pinging the sensor.

The following lines are changed:

<include file="$(find realsense2\_camera/launch/rs\_camera.launch"/>   
<include file="$(find velodyne\_pointcloud)/launch/VLP16\_points.launch">

These lines set the working directories for the ROS packages which interface with the hardware. If these are wrong, the robot cannot use the hardware. In the current version, the robot uses a realsense camera and a velodyne lidar. Above lines should work if the same hardware setup is present and the ROS packages are installed and made correctly.

**Please pay attention to the different setup variables in the exploration.launch file. If the wrong kind is selected, the program could potentially pick a wrong sensor configuration.**

<!--  
Valid robot\_config values (default value can be changed for easy launch command; more configurations can be added if needed):  
- jackal\_astra\_rplidar: Clearpath Jackal robot with front-mounted Intel camera and RPLIDAR-A1 2D laser scanner.  
- jackal\_astra\_velodyne: Clearpath Jackal robot with front-mounted Intel camera and Velodyne VLP-16 3D LiDAR.  
→  
<arg name="robot\_config" value="jackal\_astra\_rplidar"/>

This code example sets the configuration of the *Exploration.launch* file to use the intel x RP sensory setup. Like noted before, please watch the name of these when using them.

#### Validation

To validate that the lidar was integrated correctly and the exploration stack could use this lidar to execute everything it could ever want, we ran this setup in a confined space to look at the results.

A picture containing sketch, white, line, design

Description automatically generated

The program produced this map. Astounded at the new range this lidar brought with it; we could obviously assume that the robot had driven around a little bit and mapped this portion of the building.

### Robot colliding with obstacles

#### Context

The robot utilizes its lidar to spot obstacles in its surroundings. Much like how the robot sees walls, it also sees obstacles. Something called a costmap tracks which objects are where and routes the robot around these objects to avoid collisions.

#### Sub-problem

The robot bangs and stumbles around the room when driving autonomously, effectively driving into anything but free unoccupied space. Not only is this dangerous for all the expensive sensory on the robot, but the map also doesn’t really improve because of it either.

#### Goal

Make sure the jackal does not drive into any walls or objects while exploring autonomously.

#### Costmap

A very common robotics technique/data is the costmap. Quite literally to its name, it is a map of the cost (effort) the robot needs to do to get to a place. In collaboration with the lidar data, this costmap is filled with obstacles and free space. Another package, move\_base, then calculates the path to a specific location if so desired.

There is a global and a local costmap, both are identical. Objects are of course identical in all worlds or dimensions. But the robot only uses the local costmap. This local costmap however stayed empty, while the global costmap on which the local one relies; was updating properly.

#### Parameter tunability

To find the problem, I retrached every possible connection to the *costmap\_2D* package, which is the ROS package that implements the global and local costmaps. After a very long time of testing and experimenting, I found what was blocking me for so long.

The topics and obstacle height *costmap\_common\_params.yaml* weren’t configured correctly. You need root acces to find this file at the following location:

*/opt/ros/noetic/share/jackal\_navigation/params*

We initially didn’t think that topic remapping would be the problem, since topics were remapped at the very beginning of every execution; meaning that the topic essentially only was remapped once and used the same everywhere else.

Obstacle height is a tricky one to figure out, since it requires some fiddling around. If the obstacle height is too low or high in relation to your lidar height; the lidar wont register your obstacles as valid. Dismissing them, in other words. A baseline for this is that the minimum value should never reach below zero. The max height is more tricky to find out and also relies on how high or low you lidar is mounted. You should finetune this to get optimal results.

The current configuration of this file is as follows:

*‘map\_type: costmap*

*origin\_z: 0.0  
z\_resolution: 1  
z\_voxels: 2*

*obstacle\_range: 2.5  
raytrace\_range: 3.0*

*publish\_voxel\_map: true  
transform\_tolerance: 0.5  
meter\_scoring: true*

*footprint: [[-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]]  
footprint\_padding: 0.1*

*plugins:*

*- {name: obstacles\_layer, type: "costmap\_2d::ObstacleLayer"}  
- {name: inflater\_layer, type: "costmap\_2d::InflationLayer"}*

*obstacles\_layer:*

*observation\_sources: scan*

*scan: {sensor\_frame: velodyne, data\_type: LaserScan, topic: scan, marking: true, clearing: true, min\_obstacle\_height: 0.0, max\_obstacle\_height: 4.0, obstacle\_range: 2.5, raytrace\_range: 3.0}*

*inflater\_layer:*

*inflation\_radius: 0.30’*

**Please pay attention to this when upgrading or changing hardware! The above mentioned files should NEVER be altered in the current setup.**

#### Validation

To validate that the robot does not drive into any more objects again, all we have to do is destructive test its driving capabilities.

Ofcourse, this is an extreme solution. Whilst it still was tested that way, we can also visualize the costmap; more specifically: its obstacles. You can see these obstacles in the following picture.

A computer screen shot of a computer

Description automatically generated with low confidence

Everything magenta is the actual object itself, everything blue is the danger zone around this object. The robot IS allowed to be here, but REALLY does not like it to be there.

Everything that is not blue or purple, in other words: is not an obstacle; the route planner can plan around. The robot will constanly plan a route to explore more parts of a room, until it determines everything is scanned and moves on.

## Final validation

Combining all sub validations and their problems into one validation, we let the jackal move around and go about its day with the new lidar. To our delight, no new hickups or problems occurred during this event. Effectively proving that the new integration was a succes.