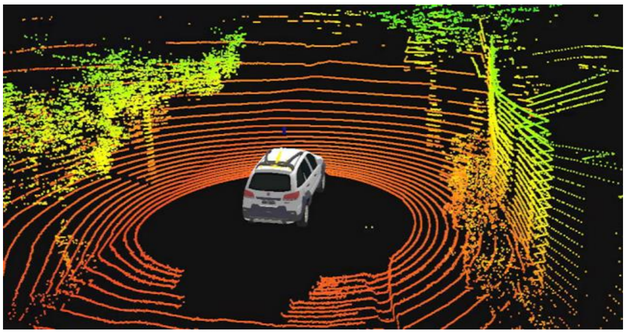
Prerequisites and preparation

In this work, the goal is to develop a C/C++ program to solve an engineering problem related to Artificial Sensor-based Perception: which finds applications in autonomous driving and mobile robotics navigation. In particular, a ground/road detection system will be developed having as input data from a 3D LiDAR sensor. The figure below shows, on the left, a point-cloud from a LiDAR (the Velodyne HDL-32 mounted on the roof of a car); on the right, the most usual Velodyne LiDAR models are depicted.



Point Cloud – the LiDAR is mounted on the car roof

LiDAR sensors from Velodyne

This practical assignment is divided in two parts. The first part involves POSIX routines, implemented in C, and the objective is to guarantee that the **threads** do not run at the same time. Therefore, it is necessary to explore and understand the concept of Synchronization services, namely **mutex** (it can be understood as a kind of binary semaphore).
The second part requires the ROS (Robot Operating System) framework, and programming languages will be C/C++.

Practical Implementation (part I)

In this assignment we will write a program (in C), following POSIX routines [Ref1], to perform **detection of ground/road points from 3D Point-clouds**. This work comprises the following tasks:

1) write a code in C to open the file (*point_cloud1.txt*), read the values from the file and pass the values to arrays/matrix inside a *struct* variable (created by you). The values, organized in columns in the file, represent the 3D Cartesian (***x,y,z***) points of a Point-Cloud from a LiDAR with 16 channels/layers (ie, from the Velodyne VLP-16 sensor). Considering the LiDAR points-values in the *struct* variable, the program has to calculate and *print*: the number of points; the minimum, the maximum and the average values of each coordinate-axis (x,y,z); and also the stand-deviation. It is recommended to create a function that receives (as one of its arguments) the file name and returns (the output) a pointer to the *struct* variable. Using the same program, repeat the process for the files: *point_cloud2.txt* and *point_cloud3.txt*.

2) create another function that takes as input a pointer to the *struct* variable from the program developed previously (item 1). This function has to perform a “pre-processing” stage, namely:
2.1) remove all the points that are located in the “back/behind” part of the car with respect to the sensor (ie, negative values of ***x***). Consider the X axis is the longitudinal axis and it is pointing ahead the car.
2.2) detect and remove two groups (clusters) of points that are located very close to the car - in the car forepart - using any technique that you think may be necessary (suggestion: a

visualization/display interface would facilitate this problem-solving: use Matlab or any software of your choice).
2.3) discard those points that clearly do not correspond to the ground/road (ie, the *Outliers*). The output of this function/routine, ie a processed point-cloud, should be write to the same *struct* variable that has been used throughout the exercise.

3) using, as input, the *struct* variable from before, developed a new function/routine to detect the points that belong to the drivable area with respect to the car, ie LIDAR points that belong to the ground/road.

4) Calculate (using *clock_gettime* based solution) and *print* the computation time associated to each of the three functions developed before. It should be guaranteed that the total computation time, considering all the *functions*, is less than 95 ms.

5) Write a program to run, in separate *threads*, each of the functions developed in the previous exercises: items 1, 2 and 3. It is mandatory to avoid all the three threads access the "*point-cloud variable*" at the same time (ie, to avoid they read-write data concurrently in the same memory location associated with the so called *struct* variable). Therefore, it is necessary to use synchronization primitives (eg, *mutex*). In other words, only one *thread* can access or write in the *struct* variable while the other threads should wait their turn to work on the *struct* variable.

Notes:
The *threads* will be activated in a sequential manner: thread#2 will process the data after thread#1 concludes its operation, and so on. The activation frequency is 10Hz (ie, a new Point-Cloud is available every 100ms) and it is the same for all *threads*.

Practical Implementation (part II)

This part of the exercise requires the ROS (Robot Operating System) framework, and programming languages will be C/C++. First, install ROS Kinetic on your Laptop. Alternatively, you can use the Desktop machines available in the Lab. It is recommended to install ROS under a LTS Ubuntu release (16.04): http://wiki.ros.org/kinetic/Installation/Ubuntu

It is only necessary the "Desktop Install" version: *sudo apt-get install ros-kinetic-desktop*
Note: for more details, see the [Ref2] **'Supp_material1_ROS.pdf'**

6) Adapt the program(s)/code(s), for the first *thread*, in order to perform the conversion of the message type *sensor_msgs::PointCloud2* (LIDAR sensor message format) to the message format *sensor_msgs::PointCloud*. Moreover, the output of the third *thread* should be published, as *geometry_msgs::PointCloud*, to a topic named "output_results": this will allow offline visualization using Rviz.

Notes:

- An example of ROS implementation is provided in [Ref2]
- See http://docs.ros.org/api/sensor_msgs/html/namespacesensor__msgs.html (convertPointCloud2ToPointCloud)
- The PointCloud (global) from the Rosbag file has to be passed to the (local) *struct* variable.

References

[Ref1] The IEEE and The Open Group. [POSIX Specification] The Open Group Base Specifications Issue 7; IEEE Std 1003.1TM-2008, 2008. [Online]. Available: http://www.opengroup.org/onlinepubs/9699919799/ .

[Ref2] L. Garrote, C. Premebida; "Robot Operating System (ROS)". Supplemental material; File: 'Supp_material1_ROS.pdf'.

[Ref3] Rui Araújo; "Sistemas de Tempo Real: Apontamentos Teóricos (Parte I)"; File: 'actstr17.pdf'.

[Ref4] Rui Araújo. "Sistemas de Tempo Real - 2014/2015". [Online]. Available: http://home.isr.uc.pt/~rui/str/str1415.html .

[Ref5] Mark Mitchell, Jeffrey Oldham, Alex Samuel. "Advanced Linux Programming." Available at: http://www.cse.hcmut.edu.vn/~hungnq/courses/nap/alp.pdf