Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Electrotécnica e de Computadores

Sistemas de Tempo Real - 2024/2025
Practical Assignment Nº2
Mutual Exclusion, Process Synchronization, Measurement of Computation Times and ROS

Introduction

In this work, the goal is to develop a C/C++ program to solve an engineering problem related to environment sensor-based perception, which finds applications in autonomous driving and mobile robotics navigation. Specifically, a ground/road detection system will be developed having as input data from a 3D LiDAR sensor. Figure 1 shows a point-cloud obtained with a LiDAR (a Velodyne HDL-32) mounted on the roof of a car (the car's image was added afterwards).

This practical assignment is divided in three parts. The first two parts involve POSIX routines, implemented in C, and the objective is to guarantee that multiple threads that perform sequential tasks (and share common global memory) do not read/write data concurrently in the same variable (memory location). Therefore, it is necessary to explore and understand the concept of synchronization services, namely **mutexes** (lock or mutual exclusion, a kind of binary semaphore). The third part requires the use of the ROS (Robot Operating System) framework, and the programming languages will be C/C++.

Practical Implementation (part I)

In this assignment we will write a program (in C), using POSIX [1] routines, to perform the **identification of ground/road points from 3D point clouds** obtained by a LiDAR. This work comprises the following tasks:

1. Implement a function that opens the file "`point_cloud1.txt`", reads the values from that file and passes the values to 3 dynamic arrays (x, y, z) inside a struct variable (created by you). The values, organized in columns in the file, represent the 3D Cartesian coordinates (x, y, z) of points of a point-cloud from a LiDAR with 16 channels/layers (acquired with a 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, the average and the standard-deviation of the values of each coordinate (x, y, z). 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, the process is to be repeated for the files "`point_cloud2.txt`" and "`point_cloud3.txt`".

2. Implement another function that takes as input a pointer to the struct variable created previously (in task 1). This function has to perform a "pre-processing" stage in order to reduce the amount of data points, specifically:

   (a) Remove all the points that are located in the "back/behind" part of the car with respect to the sensor (i.e., points with negative values of x coordinate). Consider that the X axis is the longitudinal axis of the car and that it is pointing ahead.

   (b) 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).

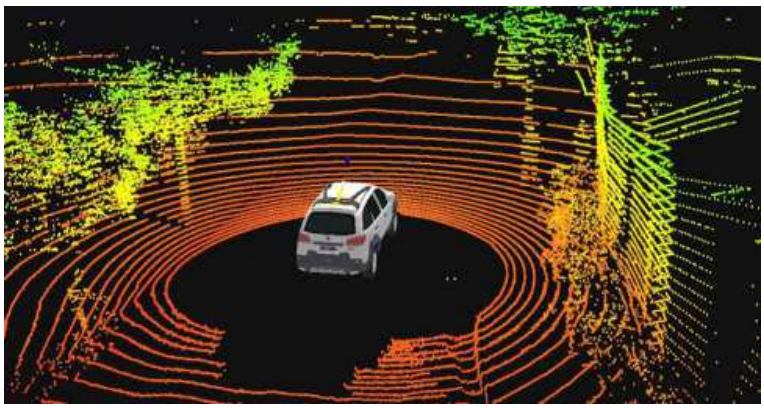

Figure 1: Point Cloud obtained by the LiDAR mounted on the car's roof.

(c) Discard also points that clearly do not correspond to the ground/road (i.e., the *outliers*). The output of this function, i.e. a processed point-cloud, can be written to the same `struct` variable that has been used throughout the exercise.

3. Using, as input, the struct variable used before, develop a new function to identify the points that belong to the drivable area with respect to the car, i.e. LIDAR points that belong to the ground road, discarding the other points (corresponding to walls, sidewalks and other obstacles).

4. Calculate (using `clock_gettime()`) 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 (as a new point cloud is generated every 100 [ms]).

Practical Implementation (part II)

5. Write a program to run, in three separate *threads*, each of the functions developed for items 1, 2 and 3. It is mandatory to avoid that all the three threads access the "*point-cloud variable*" at the same time (i.e., 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 (e.g., *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 same *struct* variable.

   Alternatively you can use three *struct* variables, each one being the output of each function. This is a typical producer-consumer problem and the three functions can run concurrently after having read the previous function output data. This solution is recommended over the first one as can run faster in a multiprocessor system. Once again the three functions need to be synchronized (you can use POSIX semaphores).

   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. Calculate and print the time between consecutive runs of the first thread. The activation frequency is 10 [Hz] (i.e., a new Point-Cloud is available every 100 [ms]) and it is the same for all threads. Alternatively you can have the threads to run as fast as they can (running permanently, waiting for their data and processing it afterwards).

Practical Implementation (part III)

This part of the exercise requires the ROS (Robot Operating System) framework, and the programming languages will be C/C++. First, install ROS Noetic 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 (use version 20.04): `http://wiki.ros.org/noetic/Installation/Ubuntu`

It is only necessary the "Desktop Install" version: `sudo apt-get install ros-noetic-desktop` . Note: for more details, see the [2] '`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 [2];
   - 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.

Report and Material to Deliver

A report should be delivered to the professor in `PDF` format. The report should contain the following information in the first page: name of the course ("disciplina"), title and number of the practical assignment, names of the students, number of the class ("turma"), number of the group. It should be submitted through the Nónio system (`https://inforestudante.uc.pt/`) area of the "Sistemas de Tempo Real" course in a single file in `zip` format that should contain all the relevant files that have been involved in the realisation of the practical assignment. The name of the submitted `zip` file should be "`tXgYrZ-str25.zip`", where "`X`", "`Y`" e "`Z`" are characters that represent the numbers of the class ("turma"), group, and practical assignment, respectively.

References

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

[2] Luís Garrote and Cristiano Premebida. Robot operating system (ROS), 2017. Supplemental material; File: 'Supp_material1_ROS.pdf'.

[3] Rui Araújo. Sistemas de Tempo Real - 2021/2022. [Online]. Available: https://ruiaraujo.rf.gd/~rui/str/.

[4] Rui Araújo. Sistemas de tempo real: Apontamentos teóricos (parte i). File: 'actstr20.pdf'; [Online]. Available: https://inforestudante.uc.pt/.

[5] Linux Kernel Documentation. Linux Kernel Documentation Index. [Online]. Available: http://www.kernel.org/doc/ .

[6] The Linux Man-Pages Project. The Linux Man-Pages Project. [Online]. Available: http://www.kernel.org/doc/man-pages/ .