Sistemas de tempo real – STR – Real-time Systems

Memo Aulas práticas / Lessons 9 – 12 (version 1)

Make groups! → Report name: tXgYr2-str2021

Practical Assignment No. 2 - support material: point_cloud1, point_cloud2, point_cloud3, Supp_material1_ROS.pdf.

All the software in this practical assignment should be developed/configured to run in more than one processor so the threads run concurrently.

Practical Assignment No. 2 (Text and comments)

Practical Implementation (part I)

Task 1 of the assignment (10%):

Implement a function that opens the file "*point_cloud1.txt*", read the values from that file and pass 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*".

Create the struct t_point_cloud, considering several alternatives:

```
typedef struct {
   double *x, *y, *z;  // coords of point i: x[i], y[i], z[i]
   int npoints;
} t_point_cloud;
```

```
typedef double Point3D[3];  // Cartesian coordinates x, y, z
typedef struct {
   Point3D *coord;        // coords of point i: coord[i, 0..2]
   int npoints;
} t_point_cloud;
```

```
#include <vector>
typedef struct {                  // dynamic vector (container)
   vector <double> x, y, z;     // point i: x[i], y[i], z[i]
} t_point_cloud;       // This type of structure may be too slow

                  // myvector.push_back(x); Add element at the end
```

Another possibility is to use a linked list class (dynamic memory).

Beware that you should choose a solution that runs fast (efficient).

If using an array, a point to be eliminated may be overwritten by the last point in the array, which is faster than moving all the remaining points one position above (assuming that the order of the points is not important). In a processing loop, this overwritten point must also be processed (the loop control variable cannot be increased).

Create a C/C++ function to read a cloud file to a struct `t_point_cloud`. First read the file to count the number of points, then allocate memory, and then read again all the points to the struct.

The struct variable should be global to be used by more than one thread.

Create a function to calculate the minimum, the maximum, the average and the standard-deviation values of each coordinate-axis (x,y,z) from a struct `t_point_cloud`.

Alternatively these values can be calculated while reading the points coordinates from the file.

Task 2 of the assignment (20%):

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:

2.1) 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 the X axis is the longitudinal axis of the car and that it is pointing ahead.

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 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.

Provide a means to visualize the point cloud of a struct `t_point_cloud` (use Matlab?). This will help you visualize the problem and the results. Report these visualizations.

Create a C function to perform the 3 sub-tasks.

Consider the car dimensions to be 4 x 2 m. Points belonging to the car are removed.

Points too far ahead (over 30 m ?) or too much to the side (over 10 m ?) should also be removed (these distances would depend on the car's velocity).

Task 3 of the assignment (20%):

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).

You may consider a grid (30 x 20 m ?) and calculate the maximum and minimum z on each element (square) of the grid. If these values are reasonable, not very high (< 1.5 m ?) nor very different (flat ground, zmax − zmin < 1 m ?) the grid element belong to the drivable area.

Another solution would be to consider the ground (and the LIDAR cloud points) divided in strips (20 cm wide?) along the x-axis. On each strip would be calculated the drivable area (length), considering the same criteria as in the solution above. Another (better) possibility is to order the points of each strip in ascending X and calculate the inclination (dz/dx) of consecutive points of the strip. Where the inclination is high (>20%?) there is an obstacle (can be a hole). The points of that strip, ahead the obstacle, should be removed (are non-drivable).

In the first algorithm, the points belonging to non-road squares are eliminated. The squares are processed sequentially. In the second algorithm, the points of a strip that are farther than an obstacle are eliminated. The strips are processed sequentially. In both algorithms the first step is to identify the points belonging to the square or strip that is being processed.

Task 4 of the assignment (10%):

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).

Write a C code to obtain (measure) the computation time of each of the 3 functions.

Check the total computation time. Do tasks meet the temporal deadline?

Task 5 of the assignment (30%):

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 *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 10Hz (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).

The threads should synchronize in a way as to allow some parallelization and avoid reading-writing data concurrently in the same variable (memory location). You may use 2 (or more) variables (structs) to communicate between the 3 threads.

See the chapter on communication and synchronization between processes in the teacher's notes (pages 38+, namely page 42).

See also:

- Synchronizing Threads with POSIX Semaphores

http://www.csc.villanova.edu/~mdamian/threads/posixsem.html

- Mutex lock for Linux Thread Synchronization

https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/

Task 6 of the assignment (10%):

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 (function convertPointCloud2ToPointCloud)**
- **The PointCloud (global) from the Rosbag file has to be passed to the (local) *struct* variable.**
- **sensor_msgs::PointCloud -> old format**
- **sensor_msgs::PointCloud2 -> new format.**

Última versão estável do ROS : Noetic Ninjemys (maio 2020, recommended for Ubuntu 20.04).

```
$ sudo apt-get install ros-noetic-desktop-full
```

Penúltima versão estável do ROS : Melodic Morenia (maio 2018, recommended for Ubuntu 18.04).

```
$ sudo apt-get install ros-melodic-desktop-full
```

Deve ser escolhida a versão estável do ROS compatível com a versão Linux instalada. Ver lista da compatibilidade das versões em:

`http://wiki.ros.org/Distributions#List_of_Distributions`

ROS Courses and materials (English):

https://www.youtube.com/playlist?list=PLRG6WP3c31_U7TFGduEIJWVtkOw6AJjFf

http://wiki.ros.org/ROS/Tutorials

ROS Tutorials.

https://discourse.ros.org/t/download-the-ros-robot-programming-book-for-free/3917

ROS book: "ROS Robot Programming, A Handbook.

http://marte.aslab.upm.es/redmine/files/dmsf/p_drone-testbed/170324115730_268_Quigley_-_Programming_Robots_with_ROS.pdf

Programming_Robots_with_ROS. A practical introduction to the robot operating system.

Read Part I. Fundamentals. Introduction, Preliminaries till roslaunch.

https://www.researchgate.net/publication/314101187_Programming_for_Robotics_-_Introduction_to_ROS

Programming for Robotics - Introduction to ROS

http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data

This tutorial will teach you how to record data from a running ROS system into a .bag file, and then to play back the data to produce similar behavior in a running system.

http://wiki.ros.org/Bags/Format/2.0

ROS bag file format.

https://www.cse.sc.edu/~jokane/agitr/agitr-letter-bag.pdf

Recording and replaying ROS messages.

http://wiki.ros.org/rosbag

The rosbag package - provides a command-line tool for working with bags as well as code APIs for reading/writing bags in C++ and Python.

(search for ROSbag)

http://wiki.ros.org/pcl/Overview

PCL (Point Cloud Library) Overview

http://docs.ros.org/melodic/api/sensor_msgs/html/msg/PointCloud.html

sensor_msgs/PointCloud Message

Clues

https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/

http://www.cplusplus.com/reference/mutex/mutex/lock/

Authentication at DEEC' Labs:

Username: your_DEEC_mail_account Password: your_password

Username (guest): aconvidado Password: cmxa5wid