Real Time Systems – 2023/2024

Practical Assignment #2

Mutual Exclusion, Process Synchronization, Measurement of Computation Times and ROS

Duarte de Sousa Cruz PL2 G5
2017264087

Gonçalo Nereu Figueiredo PL2 G5
2020226281

1 Introdução

Neste trabalho, o objetivo é desenvolver um programa C/C++ para resolver um problema de engenharia relacionado com a perceção baseada em sensores ambientais, especificamente, um sistema de deteção de solo/estrada utilizando dados de um sensor LiDAR 3D. [1]

O trabalho é dividido em três partes, onde as duas primeiras partes envolvem rotinas POSIX implementadas em C, visando garantir que várias threads que executam tarefas sequenciais não leiam/escrevam dados simultaneamente na mesma variável. A terceira parte requer a utilização da estrutura ROS (Robot Operating System) [2] e as linguagens de programação utilizadas serão C/C++. O objetivo geral é aplicar esses conceitos e tecnologias no contexto da condução autónoma e da robótica móvel navegação.

2 Exercícios propostos (Parte I)

2.1 Exercício 1

Neste exercício é nos pedido para abrir três ficheiros, que contêm valores de uma "point-cloud" de um sensor LiDAR 3D, e passar os valores para 3 arrays dinâmicos de uma struct criada por nós.

Para isto implementamos uma função chamada "structpnt". Esta função começa por ler o ficheiro dos pontos obtidos da leitura do LiDAR, passado por argumento, e passa para uma struct chamada coords. Esta struct contém 3 arrays dinâmicos x,y e z e um inteiro que contém o número total de pontos do "point-cloud". De seguida, vamos calcular os valores máximos e mínimos de cada coordenada (x,y,z). Para isto temos um loop que percorre todo o ficheiro linha por linha, fazendo uma comparação de cada coordenada com as respetivas cornadas dos pontos anteriores. Nesta secção também aproveitamos para calcular as médias, usando os somatórios das coordenadas percorridas e fazendo a divisão com o número total de pontos, guardada na struct coords. Por último, calculamos o desvio para cada coordenada x, y e z, usando a fórmula,

$$S = \sqrt{\frac{\sum(\chi_i - \bar{\chi})^2}{n-1}} \tag{1}$$

e fechamos o ficheiro, após imprimir a informação.

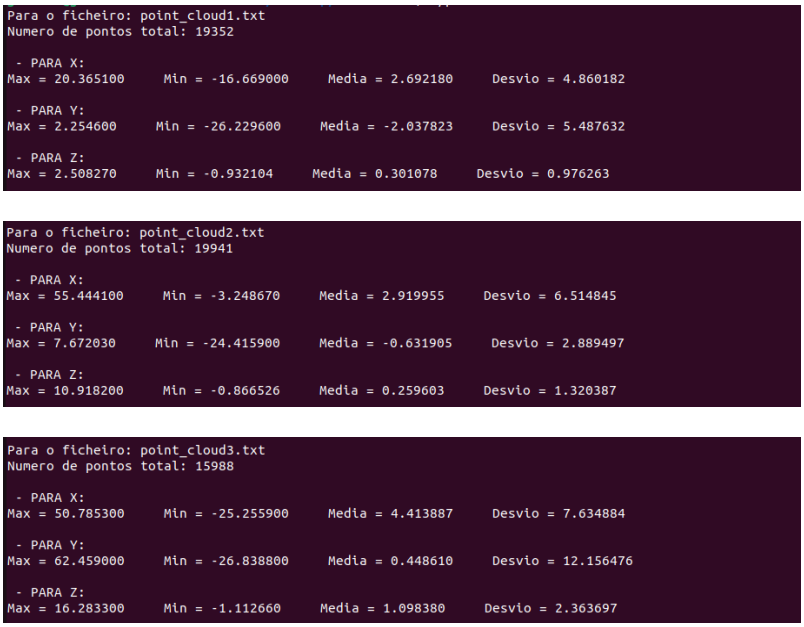


Figure 1: Número dos pontos máximos, mínimos, média e desvios, por pointcloud.

2.2 Exercício 2

Este exercício tem como finalidade reduzir o número de pontos guardados na struct "coords" do exercício anterior. Para isto, é nos pedido para remover todos os pontos localizados atrás do carro, que foi equipado com o sensor para obter a "point-cloud", os grupos de pontos demasiado próximos da parte da dianteira deste e descartar os que são "outliers".

Para isto implementamos uma função chamada "pre_processing". Esta função lê todos os pontos existentes na struct e verifica se estes se encontram nas restrições do problema. Caso consigam assegurar essas condições, esses pontos são considerados válidos e guardados em novos arrays de dados.

No fim, mudamos o destino do ponteiro da struct coords para os novos arrays e atualizamos o total de pontos existentes.

2.3 Exercício 3

Neste exercício é para desenvolver uma função que identifique os pontos correspondentes à área dirigível para o carro, ou seja, os pontos que pertencem ao chão da estrada.

Para conseguir eliminar os pontos que ficam fora das restrições do problema temos que considerar um intervalo que restringe os pontos à área pertencente à estrada. Para este efeito, consideramos os pontos que não pertencem ao intervalo de $x < 30$ e $-10 < y < 10$ e são de altura superior a 1,5 metros ou a diferença do ponto mais alto e mais baixo superior a 1 metro.

Para isto implementamos uma função chamada "discardNotDrivable". Esta função lê todos os pontos da struct e verifica as condições descritas anteriormente. No caso do ponto pertencer ao intervalo de x e y, guarda o índice deste para ser removido e determina o ponto de altura máxima e mínima.

No fim, os pontos válidos são guardados em arrays novos, criados com o uso da função "malloc()" e mudamos o ponteiro da struct inicial "coords" para estes arrays.

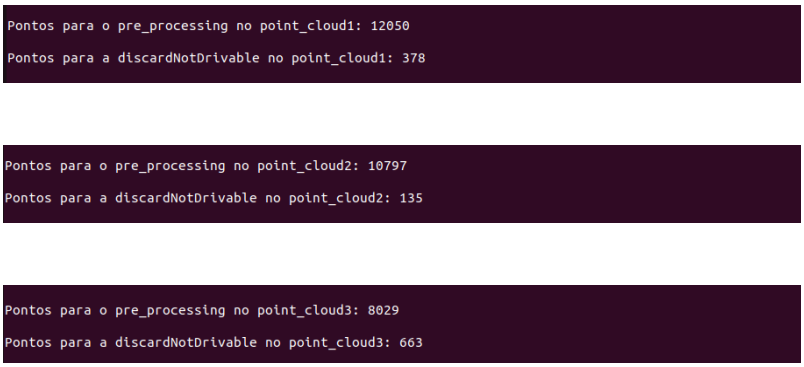


Figure 2: Número de pontos após processamento.

2.4 Exercício 4

Neste exercício é para calcular o tempo de computação das funções implementadas anteriormente e temos de garantir que é menor que 95ms. Esta implementação está no ficheiro do primeiro exercício, na função "main()".

Como podemos ver na imagem, conseguimos garantir as restrições do enunciado com sucesso.

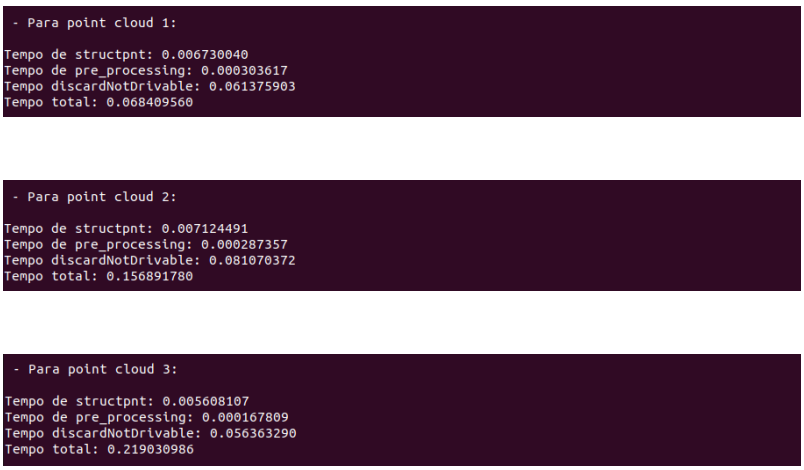


Figure 3: Tempos das funções, por pointcloud.

3 Exercícios propostos (Parte II)

3.1 Exercício 5

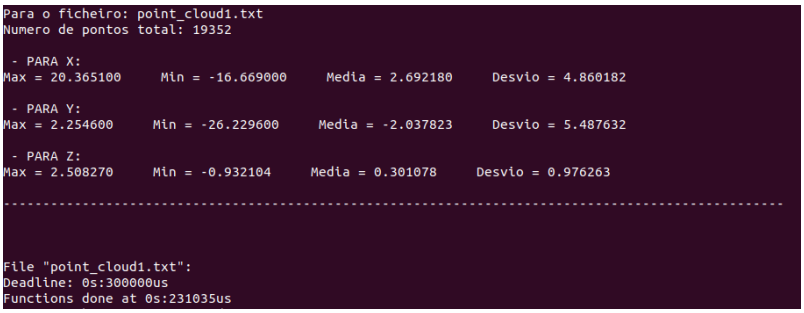
Nete exercício é para reescrever as funções dos exercícios 1, 2 e 3 em três POSIX threads separadas. É exigido que nenhuma das funções aceda à variável coords durante a sua execução. Decidiu-se, portanto, pela utilização de POSIX Semaphores para a sincronização das threads. Cada uma delas é inicializada e ficará responsável por uma das três funções executadas.

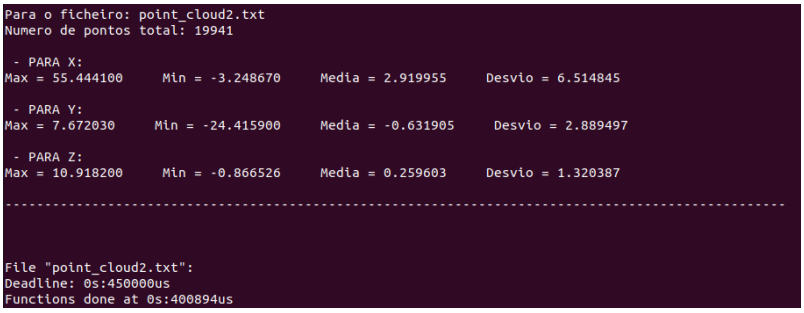
A primeira thread ficará encarregue da execução da função "thread_pointer()", que irá executar a função criada no exercício 1, a qual processa os dados pelo LiDAR nos ficheiros de texto, para uma struct. Neste caso, a função será executada três vezes, uma vez para cada ficheiro fornecido. Há ainda que implementar dois semáforos, um que espera o fim da função 3 e outro que dá "autorização" para a função 2 prosseguir execução.

Para a segunda thread, esta executará a função "thread_pre_processing()", que, por sua vez, irá executar a função criada no exercício 2. Resumidamente, esta função remove os pontos que não são de interesse para o programa. Tal como anteriormente, a função da qual a thread está responsável será executada 3 vezes, com 1 ciclo para cada struct de pontos. Nesta thread temos de recorrer novamente ao uso de dois semáforos, um que aguarda a execução da primeira função e o outro que dá a "autorização"à função 3 de executar.

O princípio da terceira thread é semelhante, executando a função "thread_discardNotDrivable()", que executa 3 vezes a função criada no exercício 3. Esta função é responsável pela remoção de todo e qualquer ponto que não pertença à estrada. Igualmente, será necessária a implementação de dois semáforos, um que aguarda o fim de execução da função 2 e outro que permite a continuação da execução da função 1.

Para este exercício foi medido o tempo de execução de um ciclo. Em cada ciclo deverá ser executado até 150 ms, uma vez que existe um problema com a meta temporal do exercício 3. Logo, a nossa frequência de execução estará nos 150 ms e não nos 100 ms.





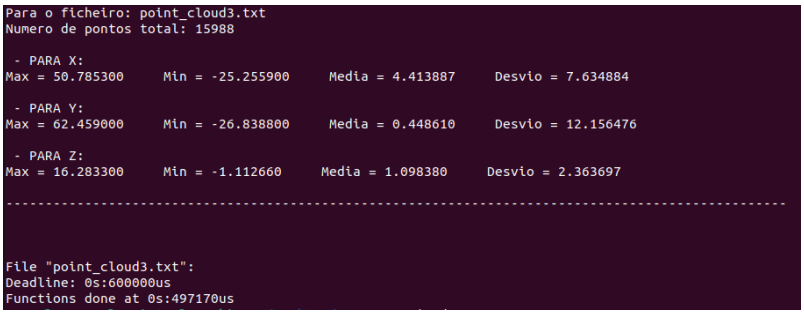


Figure 4: Tempos das threads, por pointcloud.

4 Exercícios propostos (Parte III)

4.1 Exercício 6

Neste exercício, vamos adaptar o programa para utilizar o middleware ROS(Robot Operating System) para a resolução do exercício. Para isto, tivemos que implementar um package de ROS para conter o nó, desenvolvido em C++. As instruções para criar um workspace e compilar o package, estão no ficheiro "README" que vão junto na submissão.

Em primeiro lugar, corremos a gravação do ficheiro ".bag" dado pelo professor. Nesta, conseguimos verificar qual o tópico para onde as leituras do LiDAR estão a ser transmitidas. Com isto, no nosso nó temos que subscrever ao tópico "/velodyne_points", tópico usado na gravação, para conseguir termos acesso aos valores. Esta chamada está presente na função main() do nó.

No primeiro thread f1(), vamos proceder à conversão das mensagens lidas pelo sensor "sensor_msgs::PointCloud2" para o tipo "sensor_msgs::PointCloud". Para isto, usamos uma função já incluída nas livrarias "convertPointCloud2ToPointCloud()" que faz a conversão diretamente.

No segundo thread f2(), vamos adaptar o que foi feito no exercício 2, que era pedido para remover todos os pontos localizados atrás do carro, que foi equipado com o sensor para obter a "point-cloud", os grupos de pontos demasiado próximos da parte da dianteira deste e descartar os que são "outliers". Com isto, utilizamos as mesmas restrições ao navegar por todos os pontos e guardamos os pontos válidos num vetor do tipo "geometry_msgs::Point32".

Na última thread f3(), vamos copiar os pontos válidos para mensagens do tipo "sensor_msgs::PointCloud" e publicar no tópico pedido no enunciado "/output_results" para a sua visualização a posteriori no rviz.

Por último, para facilitar correr o programa e visualizar os resultados no rviz, criamos um ficheiro ".launch" para aglomerar a chamada do nó, com a reprodução do ficheiro ".bag" e a inicialização do programa rviz.
[1]

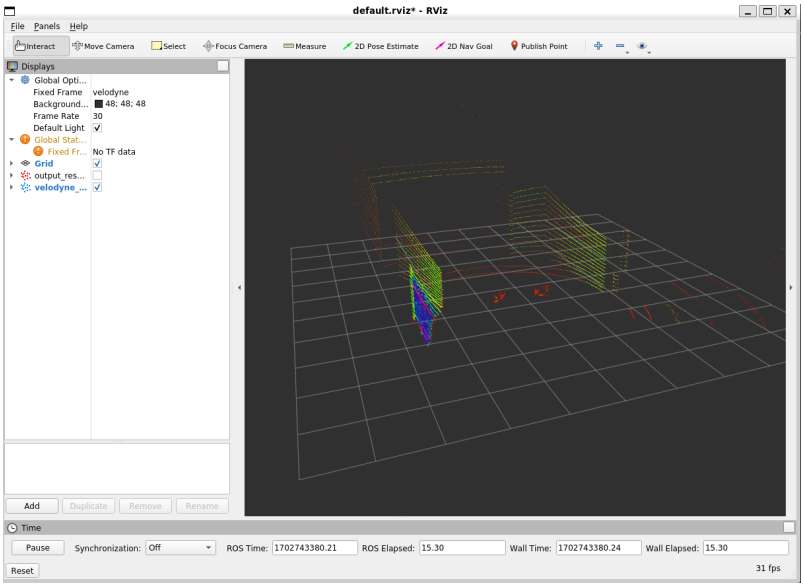


Figure 5: Pontos lidos pelos LiDAR visualizados no rviz

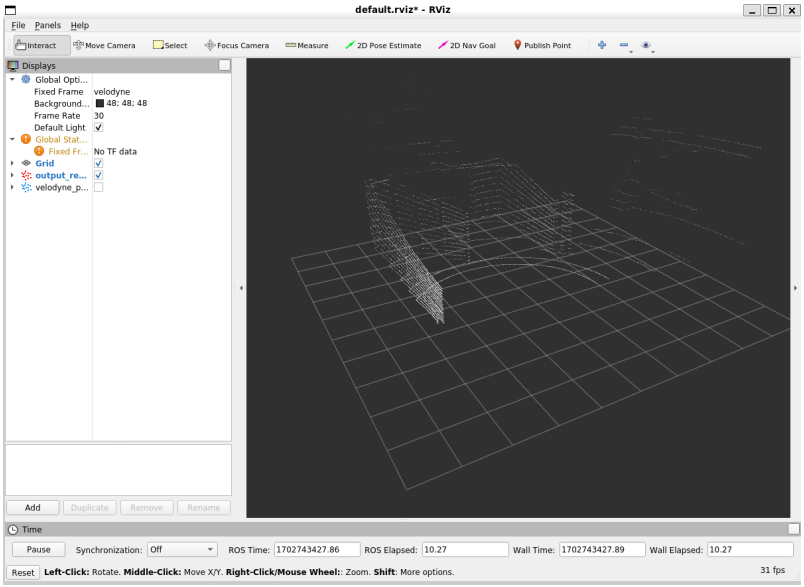


Figure 6: Pontos válidos visualizados no rviz

References

[1] Linux. The linux man-pages project.

[2] STR. *Support Material ROS*.