deer.uc
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
E DE COMPUTADORES

Sistemas de tempo Real
Trabalho Prático Nº2
Exclusão mútua, Sincronização de Processos, Medições do tempo de Computação e ROS

Leonardo Gonçalves - 2020228071 PL1
Gonçalo Bastos -2020238997 PL1

Dezembro 2023

Resumo

No âmbito do curso de Sistemas de Tempo Real foi proposto a implementação de um programa que resolva um problema de engenharia relacionado com a perceção de ambiente de um sensor, LiDAR, especificamente, na deteção de um ambiente de condução de um carro.

1 Parte I

Exercício 1 - Função `readPointCloud`

A finalidade desta função é conforme pedido ler dados de uma nuvem de pontos de um ficheiro para a estrutura de dados criada para tal, e calcular informações estatísticas sobre os pontos e colocar noutra estrutura criada para tal. As estruturas foram criadas da seguinte forma:

```
typedef struct{
    float x, y, z;
} Coord;

typedef struct{
    int point_count;
    float meanX, meanY, meanZ;
    float stddevX, stddevY, stddevZ;
    float minX, maxX;
    float minY, maxY;
    float minZ, maxZ;
} Analise;
```

Para cada ficheiro, a função começa por abrir o ficheiro e verificar se este é bem aberto, depois faz uma primeira passagem para fazer a contagem dos pontos, de forma a sabermos quanto espaço é necessário realocar para a estrutura de pontos, a alocação também é verificada, de seguida os pontos são lidos para a estrutura e é calculada as estatísticas, no fim o ficheiro é fechado.

Exercício 2 - Função `preProcessPointCloud`

Esta função recebe 2 ponteiros para ponteiros para 2 estruturas Point Cloud que no funcionamento global do programa funcionam como buffers, neste caso, a estrutura de entrada será a nuvem de pontos coletada pela função `readPointCloud` e a de saída será a nuvem pré processada. Para o pré processamento utilizamos 3 parâmetros, pontos muito afastados do veiculo serão removidos, e também pontos muito próximos, o tamanho da estrutura de saída é realocada com o tamanho reduzido e a nova contagem dos pontos é retornada como parâmetro.

Exercício 3 - Função `identifyDrivableArea`

Nesta fase vamos identificar a área condutível, para isso implementámos a função `void identifyDrivableArea`$Coord*$ $points, int*point_count$, que identifica a área condutível atualizando esta área no conjunto de pontos previamente identificados.

Esta função segue a lógica da de cima, neste caso definimos, após visualizar a nuvem de pontos, um limite para a coordenada z, e os pontos fora desse limite foram removidos.

A nuvem de pontos obtida foi a seguinte:

Exercício 4

Neste último ponto da primeira parte do trabalho fomos calcular o tempo de computação associado a cada função. E obtivemos uma média de 50ms para todas as tarefas no total, tendo assim, se verificado o objetivo de o tempo de computação total ser inferior a 95ms.

Metodologia de Teste e Visualização dos Pontos

Foram criados três arquivos distintos para teste, representando diferentes etapas do processamento de nuvem de pontos. O primeiro arquivo contém dados brutos, o segundo mostra dados pré-processados e o terceiro destaca pontos conduziveis. O MATLAB foi usado para fornecer uma perspectiva gráfica para validar os algoritmos. Os arquivos de saída foram carregados no MATLAB para plotagem e inspeção visual. A visualização ajudou na avaliação da qualidade do processamento de dados e na compreensão da distribuição espacial dos pontos.

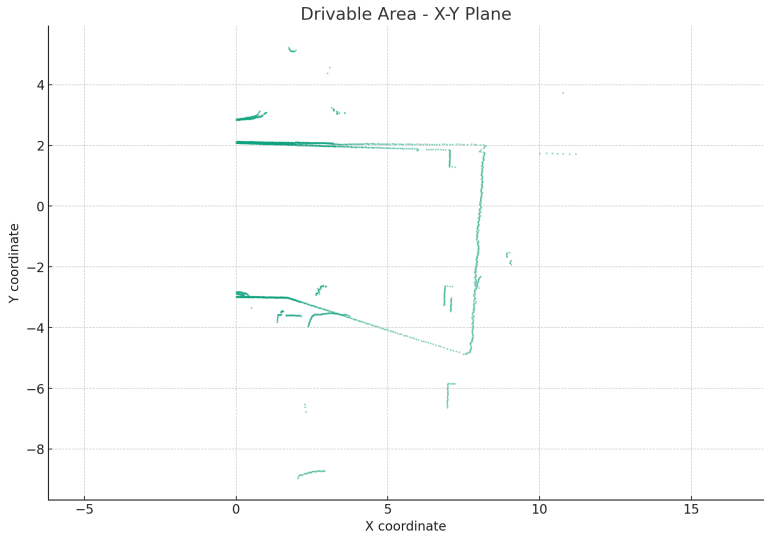

Figura 1: Drivable Area

2 Parte 2

Exercício 5

Neste exercício, modificámos o nosso programa de modo a que as três funções criadas nos exercícios anteriores fossem executadas em 3 threads separadas. Para tal, garantimos que cada thread devolvesse um buffer de saída correspondente a cada thread (*PointCloud1, PointCloud2, PointCloud3*).
De modo a garantir a sincronização das threads, para não acederem ao mesmo ficheiro ao mesmo tempo, utilizámos 2 semáforos, garantindo que a thread_2 só executa após a thread_1 ter terminado e que a thread_3 só executa após a thread_2 ter executado, para garantir este funcionamento utilizamos as funções *sem_post* e *sem_wait*.
Para além disso utilizamos o $clock_nanosleep$ para garantir que a thread_1 executa com uma frequência de 10Hz.

Fim