NavCAM-Contest

ROS-Force

6 de junho de 2021

Hugo Filipe Costa, José Nuno da Cruz Faria, Mário Jorge Pinto Cristóvão

Introdução

Este projecto foi implementado usando a infraestrutura ROS (Robotic Operative System), um plataforma robusta para o desenvolvimento de software para robôs [1]. A utilização desta plataforma permitiu-nos integrar diversos tipos de funcionalidades, sobre a mesma arquitetura, sendo que esta é de facto a derradeira vantagem da sua utilização. A esmagadora maioria dos algoritmos e códigos desenvolvidos estão em Python 3.8, o que requer instalar a versão **Noetic** da plataforma ROS, disponível para Ubuntu 20.04.

Todo o código está disponibilizado neste GitHub, assim como uma explicação mais detalhada dos diferentes módulos do sistema.

1 Características da câmara D435i

Os dados usados para cumprir estes objectivos foram gravados no recinto da ActiveSpace no dia 7/12/2020. Consistem em gravações de 5s a 30fps a diversas distâncias de uma parede ou vidro em condições de iluminação semelhantes.

[1] http://wiki.ros.org/ROS/Introduction

1.1 C1, C2 - Distancia máxima/mínima a que se detecta uma superfície opaca

Valor Real (mm)	Média (mm)	Desvio Padrão (mm)
200	181.37	0.48
1000	1038.1	6.75
3000	3153.42	78.97
5000	5322.0	231.44
7000	7706.77	723.22
9000	10017.78	732.35
10000	11452.5	1224.38

Tabela 1: Resultados da análise dos dados para parede opaca.

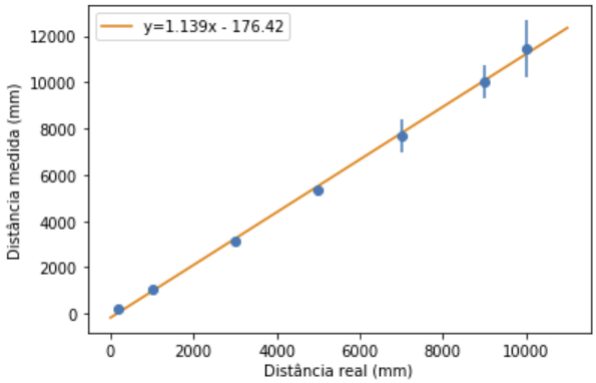

Figura 1: Reta de calibração da distância medida em função da distância real para a câmara Intel RealSense D435i.

Observa-se que os dados podem ajustar-se bem uma reta, e por isso pensamos que poderá ser possível calibrar a câmara para obter melhores resultados e até medir distâncias superiores à prevista na Datasheet. No entanto não sabemos como é que as condições de iluminação afectam o declive da reta de calibração.

1.2 C3, C4 - Distancia máxima/mínima a que se detecta um janela

Valor Real (mm)	**Média (mm)**	**Desvio Padrão (mm)**
200	217.76	15.22
1000	2425.67	19.83
2000	1562.53	1883.25
5000	23895.21	17785.4
10000	-1416.40	3784.53

Devido à natureza do sensor de distância (que é um sensor infravermelho), é muito difícil obter distâncias precisas quando existem objectos transparentes à radiação infravermelha na imagem. Nos nossos testes podemos observar que para distâncias superiores a 20cm, o erro na distância medida aumenta drasticamente, o que nos impede de detectar qualquer tipo de objecto translúcido.

1.3 C15 - Consegue detectar uma caixa de cartão a uma distância de 5m com uma precisão de 1cm?

Pelos dados obtidos nos desafios C1 e C2, observamos que a uma distância de 5m temos um erro de 23.14 cm, logo não é possível obter uma precisão de 1cm na detecção da caixa. Nos desafios

2 Detecção e *tracking* de objectos

Neste projecto adoptamos duas abordagens diferentes para a detecção e segmentação de objectos. Inicialmente, recorremos ao modelo YOLO (You Only Look Once) para efectuar a detecção de objectos usando a imagem RGB. No entanto, como estes modelos de detecção de objetos devolvem apenas uma "*bounding box* box" do objeto, é necessário outra abordagem para separar os objectos da imagem de fundo. Para tal, explorámos também o uso de um modelo de segmentação semântica designado de DeepLab. Em ambos, os modelos detectam e classificam os objectos que encontram na imagem. No entanto, o YOLO funciona com *bounding boxes* e suporta a detecção de múltiplos objectos da mesma classe, e o DeepLab funciona com um mapa de segmentação, que classifica regiões de "pixeis" de acordo com a classe do objecto que é detectado, no entanto, nesse mapa não é possível diferenciar objectos da mesma classe.

A maior vantagem do YOLO é que consegue diferenciar objectos da mesma classe mesmo estando muito próximos ou até sobrepostos, mas pode ser complicado remover o fundo presente nas *bounding boxes*.

Relativamente ao algoritmo de rastreamento de objectos, este só e aplicado às Bounding Boxes dadas pelo YOLO. O algoritmo foi baseado no algoritmo de SORT implementado pelo Alex Bewley, sendo adicionadas funcionalidades novas como a possibilidade de rastrear*bounding boxes* sobrepostas (usando a classificação do YOLO).

Os objectos depois de serem processados pelo código de tracking tem as seguintes características:

- ID (número único para cada objeto);
- Classe (Humano, semáforo, cão etc...)
- Cor (em HEX ou string se for um semáforo)
- Forma
- Coordenadas da Bounding Box
- Velocidade Vetorial Instantânea (v_x, v_y, v_z)
- Coordenadas em metros (no referencial da camera)

As provas da aplicação destes algoritmos são apresentadas nos vídeos abaixo:

1. YOLO_1, YOLO_2
2. DeepLab_1, DeepLab_2.

2.1 C5, C6, C8 - Consegue detetar um objeto a afastar-se/aproximar e obter a sua velocidade?

Sim, o sistema faz *tracking* do objecto e ao saber os tempos de chegada de 2 frames seguidos e a localização do objecto no espaço 3D é possível estimar a sua velocidade, permitindo saber se o objecto se está a afastar ou a aproximar. Este código pode ser visto no ficheiro sort_tracking.py na função *computeSpeed* da linha 148 a 171.

2.2 C7 - Consegue detetar um objecto a cair a uma distância de 2m

Sim, o nosso sistema consegue detetar um objeto a cair a uma distância de 2m. A demonstração deste desafio está neste link.

2.3 C9 - Consegue diferenciar uma pessoa imóvel de um pilar?

Sim, ambos os algoritmos de deteção de objectos conseguem diferenciar humanos de pilares, como pode observar-se aqui: YOLO_1, YOLO_2, DeepLab_1, DeepLab_1.

2.4 C10 - Qual o FOV em que consegue detetar eficazmente objectos?

A nossa solução usa a camera RGB para detectar objectos, portanto foi apenas necessário medir o FOV da lente RGB. Segundo a *datasheet* da D435i o FOV da lente RGB é de 69º graus, usando o nosso melhor modelo conseguimos detetar humanos com o FOV de 63º graus (horizontalmente).

2.5 C16 - Consegue identificar semáforos e a sua cor?

Sim, o sistema deteta e segmenta semáforos usando o algoritmo YOLO, convertendo a imagem extraida de RGB para HSV para ser mais simples de analisar. Depois de obter a imagem HSV cria 3 máscaras (vermelho, verde e amarelo) e finalmente verifica qual das máscaras possui mais valores positivos e assume que esta é a cor do semáforo. Este código pode ser visto no ficheiro sort_tracking.py na função *computeFeatures* da linha 167 a 186.

Prova em vídeo do funcionamento

2.6 C13 - Consegue determinar a forma de um objecto?

Não, apesar de termos implementado um algoritmo que tenta determinar a forma de um objecto não é robusto o suficiente. O algoritmo tenta obter as bordas de um objecto, depois transforma num polígono e conta o números de lados, se tiver mais de 5 lados é considerado um circulo.

Este algoritmo está na função *computeShape(self,thresh)* das linhas 204 a 226 do ficheiro sort_tracking.py

2.7 C17 - Consegue determinar a cor, iluminação ou sombra de um objecto?

Sim, caso o objecto não seja um semáforo, o sistema calcula o hue mais presente na imagem e a média da iluminação da imagem e publica as informações nas mensagens do objecto.

3 Estudo do IMU

3.1 C19 - Detetar inclinação do pavimento

Através do uso dos tópicos de IMU da câmara podemos tirar facilmente a orientação no formato de Quaternião. Com o uso de uma formula matricial conseguimos facilmente chegar ao valor de rotação atual em torno de qualquer um dos eixos x, y e z. Para identificar o nível de uniformidade do terreno analisamos a aceleração sentida no eixo y da câmara, variações bruscas valores são consequência de variações bruscas no terreno que a câmara percorre de momento.
Vídeo irregularidades
Vídeo orientação

3.2 C20 - Andar ao longo das paredes a uma distância constante

A aplicação é realizada de forma relativamente simples visto não estar definida uma acção directa para um robô quando a distância mínima é ultrapassada.No entanto os limites são identificados e uma mensagem de alerta é colocada no terminal.
Vídeo seguidor de paredes

4 Mapeamento e Localização

Para efetuar o mapeamento e localização (SLAM), utilizámos uma combinação de *packages* de ROS:

- *RTAB-Map* - Algoritmo de odometria visual (PnP, usando *features* GFTT e ORB), e *loop closure*, com integração no OctoMap;
- *OctoMap* - Mapeamento 3D utilizando grelhas de ocupação (*OccupancyGrid*) 3D;
- *robot_localization* - Implementação um de filtro de Kalman sem cheiro, que facilita a fusão de sensores (IMU + Odometria Visual) para a estimação do estado do sistema;

Ao longo deste projecto experimentámos diferentes algoritmos de odometria visual e diferentes implementações para mapeamento (e.g. *GMapping*[2][3]). Explorámos 2 tipos de algoritmos fornecidos pelo RTAB-Map (ou melhor, *rtabmap_ros*): Perspetive-n-Point (PnP) e Iterative Closest Point (ICP). Segundo os nossos testes, e tendo em conta os resultados dos testes C1 a C4, o ruído do sensor de profundidade da D435i inviabiliza o algoritmo ICP, dado que as nuvens de pontos são demasiado dispersas para aplicar o algoritmo. Por outro lado, usando o algoritmo PnP, obtivemos resultados muito bons, como podemos ver aqui ou aqui.

4.1 C11 - Consegue mapear uma sala 4m x 4m com pelo menos 5 objetos parados?

Sim, o nosso sistema conseguiu mapear a cave da ActiveSpace, onde observam-se diversos baldes espalhados pela sala, além de pessoas em movimento:
Demonstração mapeamento 2D
Demonstração mapeamento 3D

[2] http://wiki.ros.org/gmapping
[3] https://github.com/carlosmccosta/dynamic_robot_localization

4.2 C12 - Consegue mapear uma sala 4m x 4m com pelo menos 3 objetos parados e 2 objetos em movimento com uma velocidade de 1 ms^{-1}?

Sim, o nosso sistema conseguiu mapear a cave da ActiveSpace, onde observam-se diversos baldes espalhados pela sala, além de pessoas em movimento:
Demonstração mapeamento 2D
Demonstração mapeamento 3D

4.3 C14 - Consegue mapear uma divisão escura?

A algoritmo de odometria visual do RTAB-Map utiliza a imagem RGB para extraír os *feature points*, e como tal, não funciona bem para divisões com pouca luminosidade onde a imagem é demasiado escura para obter esses pontos. Uma abordagem que faça maioritariamente uso dos valores de profundidade (como ICP) é inviável devido ao ruído do sensor de profundidade da câmara D435i.

4.4 C18 - Consegue mapear uma divisão com iluminação altamente variável?

Não foi possível obter uma gravação com estas condições, logo não conseguimos testar o nosso sistema para este caso.

4.5 C21 - Quanto tempo demora a mapear um quarto vazio 4x4?

Não foi possível obter uma gravação com estas condições, logo não foi possível obter um tempo exato do mapeamento para um quarto vazio 4m x 4m. No entanto, pode observar-se o sistema a mapear a cave da ActiveSpace (aqui), de dimensões 30m x 15m neste link, e, de acordo com as informações do RTAB-Map, o mapa é atualizado (em média) de $0.07s$ em $0.07s$ segundos.

4.6 C22 - Quanto tempo demora a mapear 3 objectos distintos?

Se todos os objectos estiverem no FOV da câmara, assumimos que o tempo de mapear os objetos é igual ao tempo de atualização do mapa. Para o RTAB-Map, observamos que o mapa é atualizado (em média) de $0.07s$ em $0.07s$ segundos.

4.7 C23 - Quanto tempo demora a seguir uma leminscata de Gerono?

O algoritmo de odometria visual do RTAB-Map consegue atingir os 30fps, estando limitado apenas pela taxa de atualização da câmara (para uma resolução de 720p). Utilizando o filtro de Kalman mencionado anteriormente para fundir a informação do IMU com a odometria visual, obtemos uma taxa de atualização de 30 Hz ou $0.033s$ (com eventuais correções com o *loop closure* devido ao *drift* da posição com o tempo).

5 Desafio Extra 1 - Configuração da Jetson

Para facilitar a utilização da nossa solução decidiu-se portar para os dispositivos Nvidia Jetson. Criou-se uma imagem de Docker para que o sistema seja rapidamente implementado em qualquer Nvidia Jetson [4], sem nenhuma configuração adicional. Na Jetson Xavier NX, obtivemos um tempo de atualização do mapa de 0.11403s (7.645 Hz), com atualizações de odometria de 0.11403s em 0.11403s (8.77 Hz).
Neste momento as funcionalidades de detecção de objectos não estão disponíveis devido a incompatibilidade de versões, mas é possível mapear.

6 Desafio Extra 2 - Path Finding

O código consiste na implementação dos diferentes algoritmos presentes nesta biblioteca de Python3, mas escolhemos o A* como algoritmo default. A função *run(self)* executa a função *pathUpdateCallback(self)* que de segundo a segundo pede o mapa atualizado ao RTABMAP e caso o objectivo já tenha sido fornecido calcula o menor caminho desde a posição actual do robot até ao objectivo. É possível actualizar o objectivo final sempre que se deseja.

Todo o código para este desafio está neste ficheiro.

Demonstração do path finding

Referências

[1] Link para a playlist no youtube
https://youtube.com/playlist?list=PLKjkjxSZEaG09UDYbf1zcEqd3fFujIFYf

[4] Não é possível garantir funcionalidade nos modelos com RAM inferior a 8GB