Using cameras in UNEXMIN AUV for determination of the relative motion to an unstructured environment

1181153 - RODRIGO BRANQUINHO

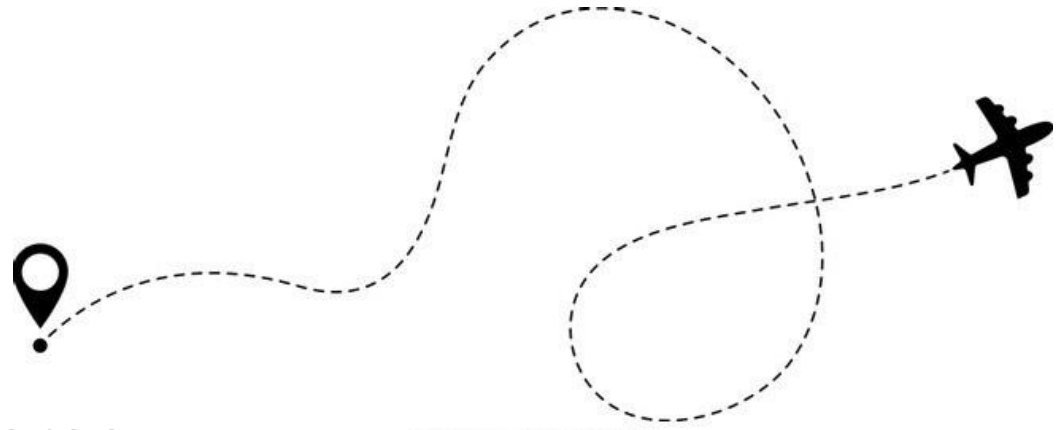
1181186 - JOSÉ SOARES

Sumário

- Introdução – Odometria Visual
- Objetivos
- Procedimento e Algoritmo
- Resultados Obtidos
- Trabalho Futuro

Introdução

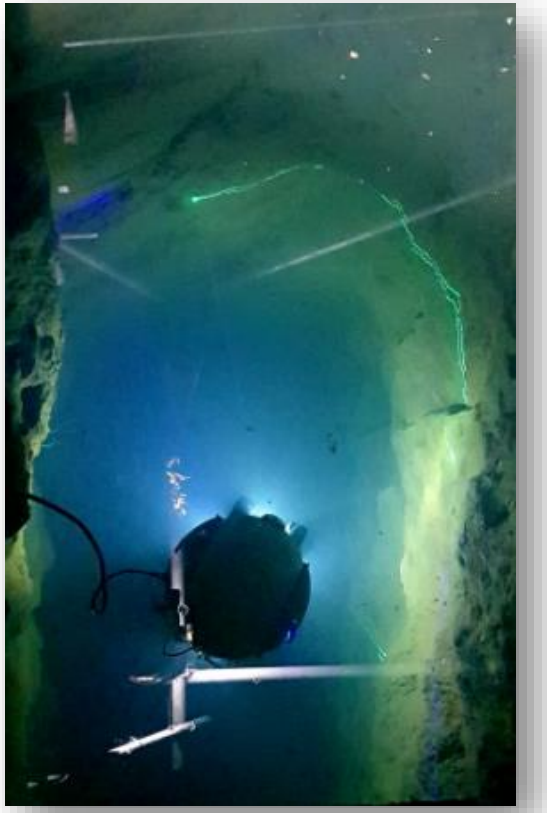
- Odometria visual: processo que determina a localização e orientação de uma câmara através da análise de uma sequência de imagens (*frames*)
- Exemplos de utilização:
  - Robôs móveis
  - Carros autónomos (*self-driving*)
  - UAVs (*unmanned aerial vehicles*)



- Odometria visual monocular *vs* stereo

Objetivos

- Processo utilizado: odometria visual monocular
- Utilização da câmara frontal
- Técnicas de processamento de imagem:
  - Definição dos parâmetros intrínsecos da câmara
  - SURF: deteção, seleção e *matching* de *features* entre *frames*
  - Estimação da *pose* da câmara ao longo dos *frames*
  - Determinação da trajetória da câmara



Procedimento Prévio

- Extração dos vídeos no Linux
  - `rosbag play bag_name.bag`
  - `rosrun image_view video_recorder _filename:=“cam_bot.avi” image:=/cam_bot/led`

```
jose@jose:~/TAVRU_KOBANYA$ rostopic list
/cam_bot/led
/cam_front/led
/cam_right/led
/clock
/nav_neo
/rosout
/rosout_agg
```

Procedimento Prévio

- Fragmentação do vídeo em *frames (.jpg)*
  - É extraído um *frame* a cada 5, de maneira a reduzir o tempo computacional

```
clear all; close all;

%% Import the video file
obj = VideoReader('Videos/cam_front.avi');
vid = read(obj);
frames = obj.NumberOfFrames;

%% Get the frames and save the images (every 5 frames)
cd frames_front
for i = 1 : 5 : frames
    % Image name
    number = sprintf('%04d', i);
    fileName = strcat(number, '.jpg');

    % Exporting the frames
    Image = vid(:, :, :, i);
    imwrite(Image, fileName);
end
cd ..
```

Algoritmo

1. Definição dos parâmetros intrínsecos da câmara frontal

$$Radial\ Distortion = [-0{,}338651 \quad 0{,}142042 \quad -0{,}030265]$$

$$Tangential\ Distortion = [-0{,}000232 \quad 0{,}000173]$$

$$Intrinsic\ Matrix = \begin{bmatrix} 1110{,}344126 & -0{,}598042 & 1039{,}193868 \\ 0 & 1107{,}604563 & 796{,}997243 \\ 0 & 0 & 1 \end{bmatrix}$$

Algoritmo

2. Ler o primeiro *frame*, converter para *grayscale* e fazer *undistort*
3. Detetar e extrair as *features* do primeiro *frame*
4. Ler o segundo *frame*, converter para *grayscale* e fazer *undistort*
5. *helperDetectAndMatchFeatures*: detetar, extrair e dar *match* das *features* entre o *frame* atual e o anterior
6. *helperEstimateRelativePose*: receber os *matched points* e estimar a *pose* da câmara no segundo *frame* relativamente ao primeiro (estimação da matriz essencial e decomposição em localização e orientação)

Algoritmo

7. Ler o *frame* seguinte, converter para *grayscale* e fazer *undistort*
8. *helperDetectAndMatchFeatures*: detetar, extrair e dar *match* das *features* entre o *frame* atual e o anterior
9. *helperFindEpipolarInliers:* retornar os índices dos *matched points* que cumprem a restrição epipolar e eliminar os restantes
10. *helperFind3Dto2DCorrespondences*: triangular os *matched points* dos 2 *frames* anteriores e encontrar os pontos correspondentes no *frame* atual

Algoritmo

11. *estimateWorldCameraPose*: estimar a *pose* da câmara para a *view* atual
12. Realizar um *bundle adjustment* para melhorar as *poses* da câmara (*refine*)
13. *helperUpdateCameraPlots*: atualizar e mover a câmara no *plot*
14. *helperUpdateCameraTrajectory*: fazer o *plot* da trajetória da câmara
15. Repetir para os restantes *frames*

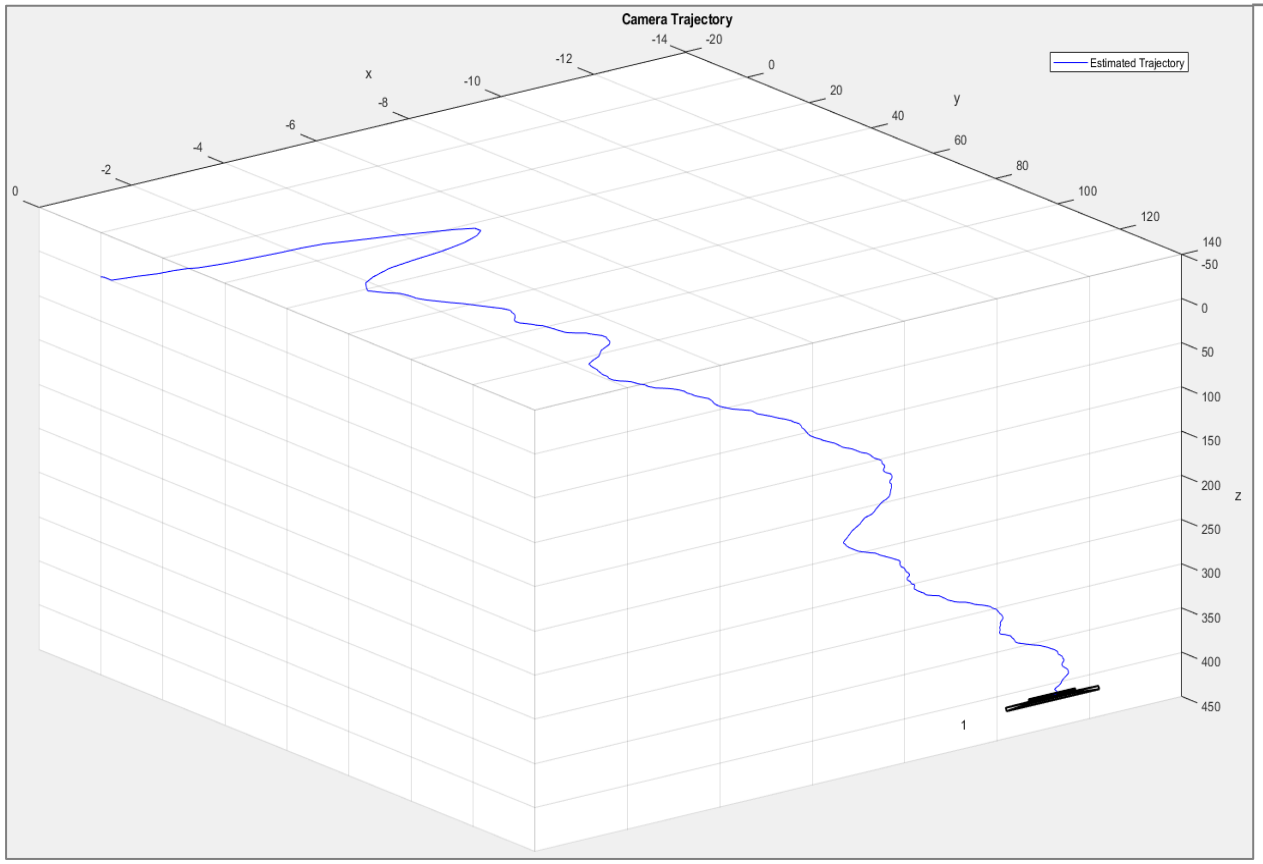
Global Bundle Adjustment

- A estimação das *poses* da câmara resultam em erros que são acumulados ao longo do tempo (efeito chamado de *drift*)
- Para reduzir este *drift*, faz-se um *refine* sequencial de todas as *poses* obtidas anteriormente, utilizando-se um *global bundle adjustment*
- No entanto, a cada novo *frame*, o tempo computacional para dar *refine* de todas as *poses* aumenta significativamente

Windowed Bundle Adjustment

- Para reduzir este tempo computacional utiliza-se um *windowed adjustment* que, a partir do 30º *frame*, melhora apenas as *poses* dos últimos 15 *frames*
- De forma a reduzir ainda mais o tempo computacional, este *adjustment* é apenas chamado a cada 5 *frames* novos

Resultados Obtidos

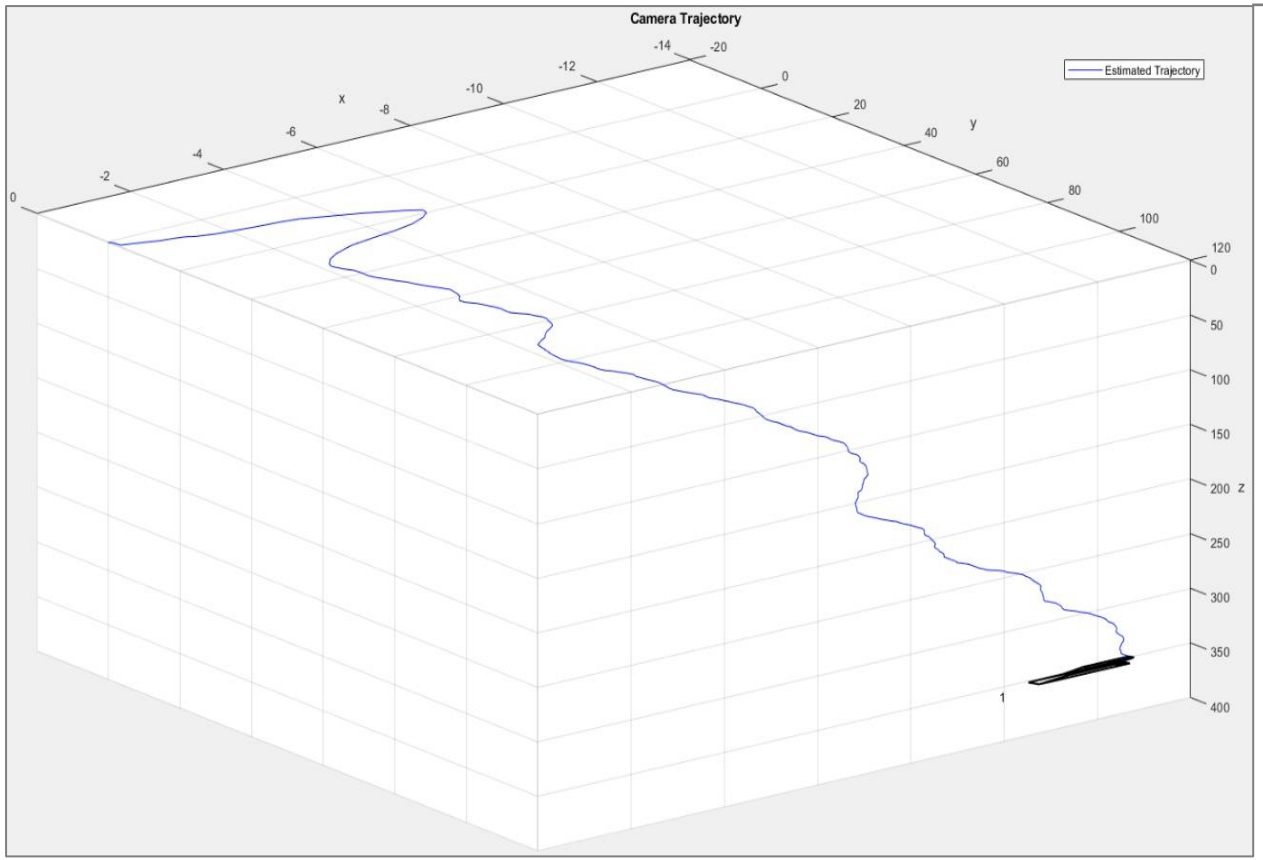


Global Adjustment

Otimização de todas as *poses* da trajetória a cada novo *frame*

Tempo Computacional:
cerca de 34 minutos

Resultados Obtidos



Windowed Adjustment

Otimização apenas das últimas 15 *poses* a cada 5 *frames*

Tempo Computacional: cerca de 2 minutos

Trabalho Futuro

- Otimização do código de maneira a reduzir o custo computacional, mantendo a sua performance
- Adicionar uma *ground truth* para obter um *scale factor*
- Utilização de mais câmaras (*stereo system*)

Referências

- http://wiki.ros.org/rosbag/Commandline
- http://wiki.ros.org/image_view
- http://avisingh599.github.io/vision/visual-odometry-full/
- https://www.mathworks.com/help/vision/ug/monocular-visual-odometry.html
- http://www.cs.toronto.edu/~urtasun/courses/CSC2541/03_odometry.pdf

Obrigado pela atenção!