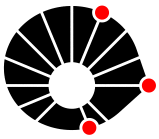

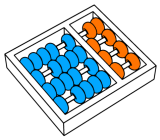

Odometria visual para robôs

W. N. Ikedo *E. L. Colombini*

Relatório Técnico - IC-PFG-17-19
Projeto Final de Graduação
2017 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Odometria visual para robôs

William Nobuaki Ikedo* Esther Luna Colombini†

Resumo

Para que um robô atue de forma autônoma, é fundamental que consiga se localizar em relação ao ambiente, e assim determinar a trajetória que deve seguir para executar sua tarefa. Nesse contexto, a odometria visual apresenta-se como um dos métodos mais utilizados, e portanto o desenvolvimento e avaliação de algoritmos precisos e robustos para esse fim são de grande relevância. Assim, este trabalho investiga o método *Direct Sparse Odometry (DSO)* aplicado ao contexto de robôs móveis. Para isso, foram realizados testes utilizando-se um modelo do robô diferencial *Robotino* no ambiente de simulação *V-Rep*, empregando-se diferentes velocidades para o robô e texturas aplicadas ao ambiente. Os resultados obtidos neste estudo colocam em questão a precisão e robustez demonstradas na publicação original, e reforçam a necessidade de maior investigação sobre as limitações do método.

1 Introdução

1.1 Odometria Visual

A odometria visual é um método que permite a um agente estimar sua localização espacial (pose) utilizando somente informações visuais obtidas por câmeras. Com os recentes avanços na área de veículos autônomos e robótica móvel, o desenvolvimento de algoritmos precisos e robustos de odometria visual ganha relevância, uma vez que tais algoritmos permitem estimar a trajetória do agente, auxiliando em sua navegação.

A utilização do odometria visual em aplicações como robôs apresenta-se como um opção atraente em relação a outros métodos de odometria por razões como o preço, acessibilidade, acurácia, e por não sofrer de fraquezas dos demais métodos, como o drift com encoders de roda, ou a dependência de sinais externos, como é o caso de métodos baseados em GPS [4].

Os métodos existentes de odometria visual variam de acordo com a forma como capturam e tratam a informação visual.

1.2 Monocular x Stereo

Existem 3 métodos para captura de imagem: Monocular, Stereo e Omnidirecional, sendo os dois primeiros os mais comuns. No caso monocular, apenas uma câmera é utilizada, obtendo-se uma opção mais barata, simples e compacta; no entanto, tal configuração sofre

*Graduando, Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP.
†Professora Doutora, Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP.

com a incerteza de escala devido a impossibilidade de se obter informação de profundidade. Tal desvantagem é compensanda quando se emprega duas câmeras (Stereo), que por outro lado apresenta maior custo e complexidade de calibração.

1.3 Método Indireto x Método Direto

Uma outra classificação ligada a odometria visual é a referente a abordagem de processamento da informação. Para se obter uma estimativa de pose e movimento no espaço tridimensional por meio de imagens, procura-se computar um estimador X por meio de um modelo probabilístico que recebe as medidas Y, extraídas das imagens de entrada [1]. Nesse contexto, o chamado método indireto primeiro pré-processa a imagem recebida pelo sensor da câmera gerando medidas intermediárias, que são então usadas pelo modelo probabilístico como entrada para fazer a estimativa. Como as medidas intermediárias feitas são medidas geométricas, como pontos e vetores, diz-se que o método indireto trabalha com a otimizaçÃo do erro geométrico; já o chamado método direto utiliza diretamente as informações vindas da imagem da câmera, e diz-se que realiza uma otimização fotométrica.

1.4 Método Esparso x Método Denso

Por fim, a classificação em métodos esparsos e densos diz respeito à quantidade de pontos amostrados. De maneira geral, métodos esparsos selecionam uma quantia limitada de pontos chave na imagem sem estabalecer uma relação entre eles, isto é, cada ponto é tratado como uma informação independente; enquanto métodos densos tentam recuperar o máximo de pontos da imagem, estabelecendo relações entre eles.

2 Direct Sparse Odometry

Direct Sparse Odometry (DSO) é um método proposto recentemete por *Engel et al* [1] que chamou a atenção por seu desempenho nos datasets utilizados nos testes de sua publicação. A primeira versão do método, publicada em 2016, é a explorada neste trabalho. Na publicação, foi apresentado um método de odometria visual utilizando a configuração monocular, direta e esparsa, combinação até então pouco explorada, que aparenta ser ideal para aplicações de menor porte que necessitassem de odometria em tempo real, como é o caso de muitos robôs.

Como um método direto, o DSO atua por meio da otimização contínua do erro fotométrico em uma janela de frames recebidos como entrada. No entanto, diferente de métodos diretos convencionais, os autores decidiram por realizaram a otimização sobre todos os parâmetros envolvimentos conjuntamente. Outra diferenciação apontada pelos autores é a de que o método emprega a calibração fotométrica além da calibração geométrica, o que segundo eles aumenta a robustez e acurácia. Vale aqui a observação de que como os testes realizados para o presente estudo foram realizados em ambiente de simulação virtual, assumiu-se que a câmera teria um comportamento ideal, e portanto a calibração geométrica foi feita assumindo-se o modelo pin-hole ideal, e julgou-se que a calibração fotométrica não seria necessária.

2.1 Formulação do Modelo

$$E_{pj} := \sum_{p \in \mathcal{N}_p} w_p \| (I_j[p'] - b_j) - \frac{t_j e^{a_j}}{t_i e^{a_i}} (I_i[p] - b_i) \|_\gamma \tag{1}$$

$$p' := \prod_c (R \prod_c^{-1} (p, d_p) + t) \tag{2}$$

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} := T_j T_i^{-1} \tag{3}$$

$$w_p := \frac{c^2}{c^2 + \|\nabla I_i(p)\|_2^2} \tag{4}$$

$$E_{photo} := \sum_{i \in \mathcal{F}} \sum_{p \in \mathcal{P}_i} \sum_{j \in_{obs}(p)} E_{pj} \tag{5}$$

O modelo probabilístico a ser minimizado é definido pelas equações acima. A Equação 1 define o erro fotométrico de um ponto p num frame de referência I_i, observado num frame I_j como a soma ponderada de diferenças quadráticas numa pequena vizinha de pixels (8, no caso da publicação). $\mathcal{N}_p$ representa o conjunto de pontos incluídos na soma, t_i e t_j são os tempos de exposição dos frames I_i e I_j, e $\|.\|\gamma$ é a norma de Huber. O valor p' é a projeção da posição de p com a profundidade inversa d_p, dado pela equação 2. O peso aplicado ao somatório é dado por 4. Finalmente, o erro fotométrico total sobre todos os pontos é dado pela Equação 5.

2.2 Seleção de Pontos Chave

Para captura de pontos chave nas imagens, o algoritmo mantém um número fixo ($N_p = 2000$ na publicação) de pontos ativos, distribuídos de forma uniforme na imagem de entrada. A seleção de pontos então segue as etapas:

2.2.1 Seleção de Pontos Candidatos

A cada novo frame, novos pontos candidatos são selecionados, levando-se em conta a distribuição na imagem, e a magnitude de gradiente em relação a vizinhança.

2.2.2 Tracking de Pontos Candidatos

Pontos candidatos são acompanhados nos frames subsequentes, para os quais são calculados valores iniciais de profundidade.

2.2.3 Ativação de Pontos Candidatos

Quando uma certa quantidade de pontos é eliminada do frame, pontos candidatos são escolhidos para repô-los. Priorizando a distribuição uniforme ao longo das imagens, os pontos selecionados serão aqueles que estiverem à maior distância dos pontos ativos remanescentes.

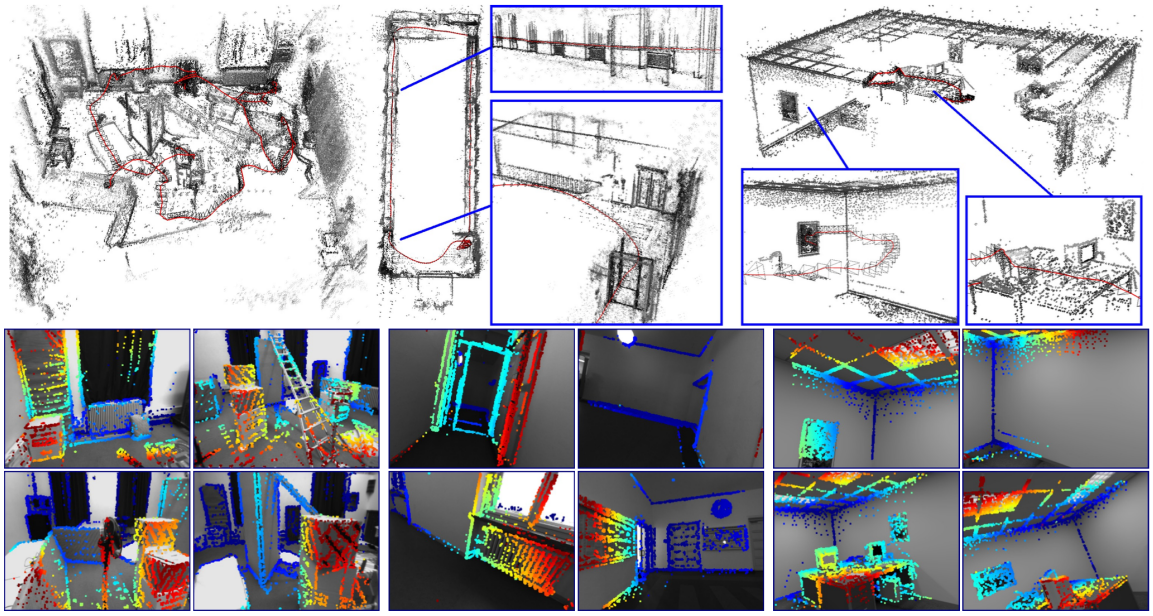

Figura 1: Resultados exibidos no artigo original indicam uma performance excelente do algoritmo, mostrando-se robusto até em cenas com pouca textura

2.3 Resultados apresentados

A avaliação do algoritmo foi feita sobre os resultados obtidos executando-se o DSO em 3 datasets (*TUM monoVO dataset*, *EuRoC MAV dataset* e *ICL-NUIM dataset*) e aponta para a superioridade do método em relação aos métodos indiretos com título de estado-da-arte atuais, apresentando performance superior, por exemplo, em cenários com pouca textura. Os autores observam, no entanto, que em situações de muito ruído geométrico nas imagens, os métodos indiretos mostram-se superiores.

Qualitativamente, os resultados obtidos pelo algoritmo de fato parecem apresentar grande precisão e robustez a variações como textura, iluminação e velocidade, como observado na Figura 1, e no video de demonstração disponibilizado na página dos autores [13].

3 Metodologia

3.1 Arquitetura

Visando criar um ambiente que permitisse a flexibilidade de realizar os mesmos testes com o simulador e com o robô real, optou-se por separar o controle do robo em um nó externo ao simulador e utilizar o sistema operacional de robótica ROS [2] como interface de comunicação entre o nó de controle e o simulador. Dessa forma, caso se deseje utilizar o nó de controle

com o robô real basta conectá-lo à interface do ROS no lugar do simulador. Seguindo a mesma decisão de design, o algoritmo do DSO também possui um nó separado que foi implementado sobre o modelo disponibilizado pelos próprios autores do método.

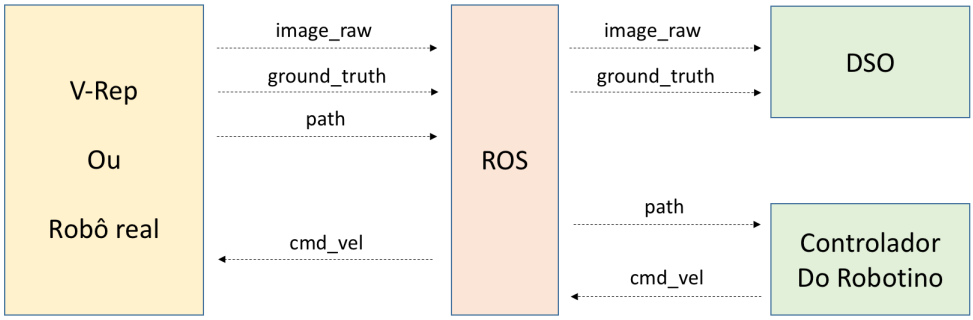

Figura 2: Arquitetura elaborada para execução dos testes

As portas de comunicação utilizadas pela interface do ROS, chamadas "tópicos", são ilustrados na Figura 2 e descritas a seguir:

- *image_raw* : input recebido pela câmera do robô;
- *ground_truth* : informações de posição e orientação absolutas do robô;
- *path* : informações do caminho a ser seguido pelo robô no simulador;
- *cmd_vel* : informações de velocidades (linear e angular) fornecidas ao robô;

3.2 Modelo e Ambiente no Simulador

A Figura 3 apresenta o modelo tridimensional do Robotino [12] e a cena criada para testes no V-Rep. O modelo possui uma câmera acoplada que foi ajustada para a resolução de 640x480 pixels. A locomoção do robô se dá por meio de três rodas na parte inferior. Um método criado no script interno do robô no simulador abstrai o controle e coordenação das rodas, de modo que apenas é necessário fornecer as velocidades lineares nas direções X e Y e a velocidade angular no plano XY.

A cena gerada para os testes consiste em uma área quadrangular com paredes, de forma a criar um caminho único para o robô seguir. Para fornecer informações visuais adicionais e para testar a robustez do algoritmo foram utilizadas duas texturas distintas, uma com padrão de mosaicos triangulares, e o outra mais limpa, com padrões retangulares mais esparsos. O chão utilizado na cena é o padrão do simulador, com textura com padrões quadrangulares.

Visto que o ambiente do simulador é ideal, assumiu-se que a câmera do robô funciona como um global-shutter, adotou-se uma calibração geométrica utilizando o modelo pin-hole, e desprezou-se a calibração fotométrica.

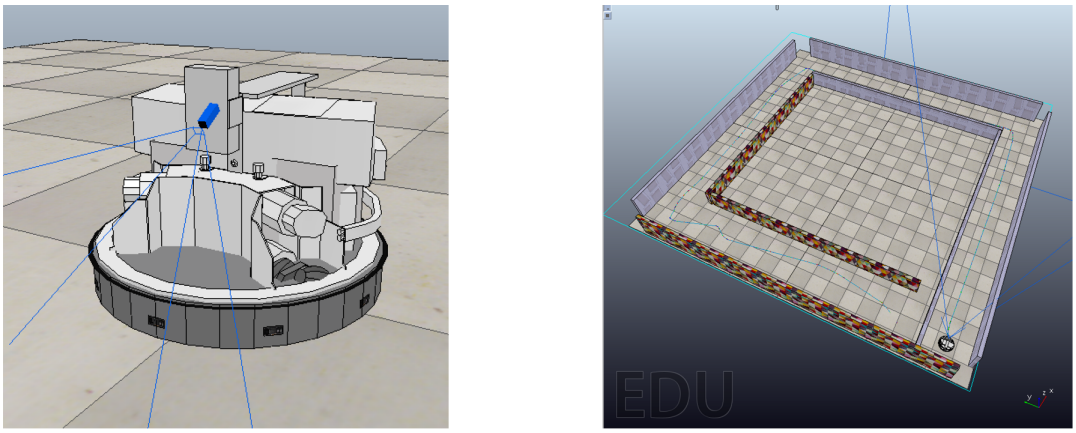

Figura 3: Modelo 3D do robotino e cena gerada para testes

3.3 Configurações

Os testes foram executados em ambiente Linux (Ubuntu 16.4) em máquina virtual (VMware Fusion) dentro do MacOs Sierra. O hardware utilizado foi um Macbook Pro 2014, com Core i7 (3.0GHz, 2 núcleos e 4 threads) e 16GB de RAM. A versão utilizada do V-Rep foi a 3.4 (rev 17), e a do ROS foi a 1.12.7 (Kinect Kame). Tanto o nó do DSO, quanto o de controle do robotino foram implementados em C++.

3.4 Procedimento de Teste

Os testes realizados consistiam em programar o robô para que este seguisse uma trajetória que percorresse os corredores externos da cena partindo de uma ponto inicial e terminasse num final, ambos pré-definidos (Figura 5), e coletar as informações de posição (x e y) e velocidade angular do *ground truth* (posição real do modelo) e das informações de odometria geradas pelo DSO para posteriormente compará-las. Para essa tarefa, utilizou-se ferramenta de Path Planning do V-Rep que, dado um ponto inicial e um final, calcula o caminho do primeiro ao segundo. Parâmetros da ferramenta para evitar colisões e curvas muita abruptas foram ajustados. O Path Planner executa sempre no início dos testes, e portanto a cada execução um novo caminho é calculado. Uma vez gerado o caminho, o robô passa a seguir um objeto de referência (dummy) que anda sobre o caminho. Com o objetivo de avaliar a performance do algoritmo a diferentes texturas e velocidades, foram elaborados testes combinando 3 velocidades distintas (0.1, 0.2, 0.4), em m/s, e duas texturas distintas (Figura 4), gerando um total de 6 cenários distintos. A partir dos resultados de ground truth e odometria coletados, foi gerado, para cada cena, um gráfico XY comparando a posição real com a estimada pela odometria.

Figura 4: Texturas utilizadas: mosaico e molduras

4 Resultados e Discussão

Os resultados obtidos nos 6 cenários de teste são apresentados nas figuras 6, 7 e 8. Para cada figura, o gráfico XY da esquerda representa a comparação entre as posições do ground truth e da odometria empregando-se a textura de mosaico, enquanto o da direita representa os resultados com a textura de molduras.

A primeira observação relevante a se fazer, qualitativamente, a respeito dos resultados apresentados é que em nenhum dos casos o algoritmo do DSO foi capaz de fazer uma boa estimativa da trajetória, o que é uma surpresa, visto que a publicação original demonstrou resultados excelentes nos datasets utilizados. Um padrão comum a todos os casos de teste é que inicialmente, com a trajetória basicamente em linha reta, o método aparenta se comportar corretamente, mas se perde ao fazer a primeira curva. Em alguns casos, como para v = 0.1m/s e textura em mosaico, percebe-se que mesmo se perdendo após a primeira curva, o algoritmo ainda consegue estimar uma certa mudança de direção, mas percebe-se que já não está mais acompanhando a trajetória. Outro fato que chama atenção diz respeito às proporções estimadas pelo método: no geral, nota-se que a odometria estima uma distância de cerca de metade da real ao percorrer o primeiro corredor. Este fato se dá por uma característica dos algoritmos de visão monocular chamado de ambiguidade de escala. De fato, uma vez que apenas uma câmera está sendo aplicada para a captura de imagem, não se tem a noção de profundidade. A Figura 9 , que mostra a reconstrução tridimensional feita pelo próprio algoritmo, reforça tais observações. Para resolver tal problema, um sensor adicional capaz de computar a escala é necessário.

Apesar de um resultado geral fraco, é possível notar alguns padrões nas diferentes configurações de cenas de teste. Em primeiro lugar, não é possível perceber nenhuma correlação da velocidade com a precisão, visto que o resultado para diferentes velocidades não foi significativamente diferente, com exceção talvez da configuração com v = 0.4m/s e textura de molduras. Por outro lado, é possível realizar algumas comparações dos resultados das duas texturas. De forma geral, a estimativa feita com a textura em mosaico foi ligeiramente

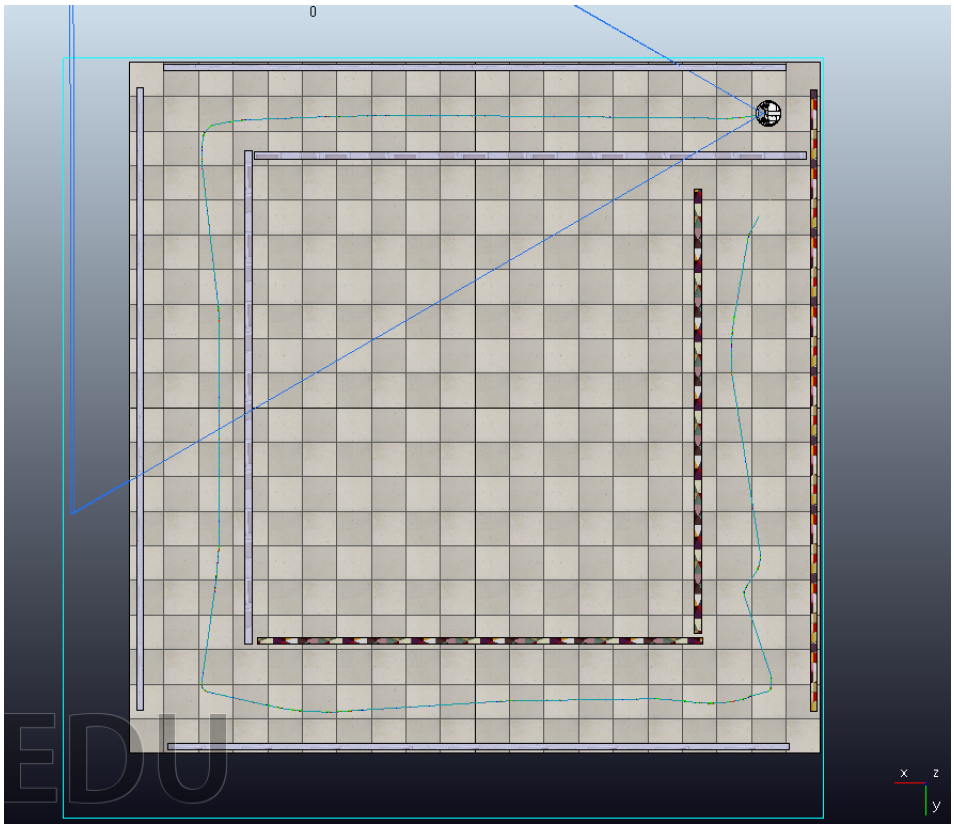

Figura 5: Exemplo de caminho gerado pelo Path Planner do V-Rep

superior que aquela feita com a textura de molduras. Soma-se a isso o fato de que para a textura em moldura, nos casos em que v = 0.2m/s e v = 0.4m/s o nó do algoritmo terminou antes do fim da simulação devido a um erro interno, o que pode ser observado pelo ground truth incompleto nesses casos. Uma possibilidade é o fato da textura de mosaico ser mais segmentada auxilie o algoritmo na reconstrução de keypoints.

Diante da discrepância dos resultados relativos à performance do DSO apresentados por este estudo em relação ao da publicação original, algumas possíveis explicações foram levantadas. Em primeiro lugar, tem-se a possibilidade de erro experimental, seja na forma como foi calibrado o método, ou estruturados os testes. Soma-se a isso a possível influência do fato de se ter executado os testes em ambiente de máquina virtual, sendo que se observou que a taxa de frames for segundo durante a execução dos testes ficou extremamente baixa (aprox. 14fps, em média). Por outra lado, não se pode descartar a possibilidade de, de fato, ter-se encontrado uma fraqueza do algoritmo. Os testes realizados pelos autores, apesar de terem sido feitos em diversos datasets, não cobrem todos as situações possíveis a que o algoritmo poderia ser exposto. Visto que o método é recente, e ainda não há outros estudos publicados até o momento que façam uma avaliação mais formal do DSO, reforça-se aqui a importância desse tipo de estudo.

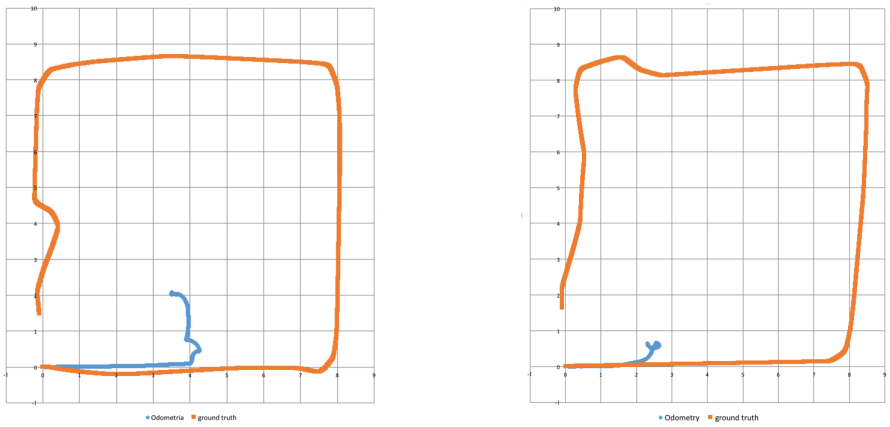

Figura 6: Comparação entre ground truth (laranja) e odometria (azul) para velocidade 0.1m/s.

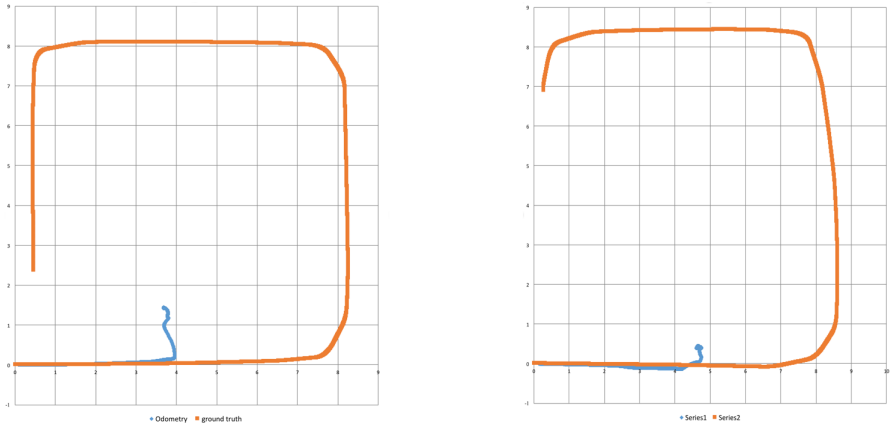

Figura 7: Comparação entre ground truth (laranja) e odometria (azul) para velocidade 0.2m/s.

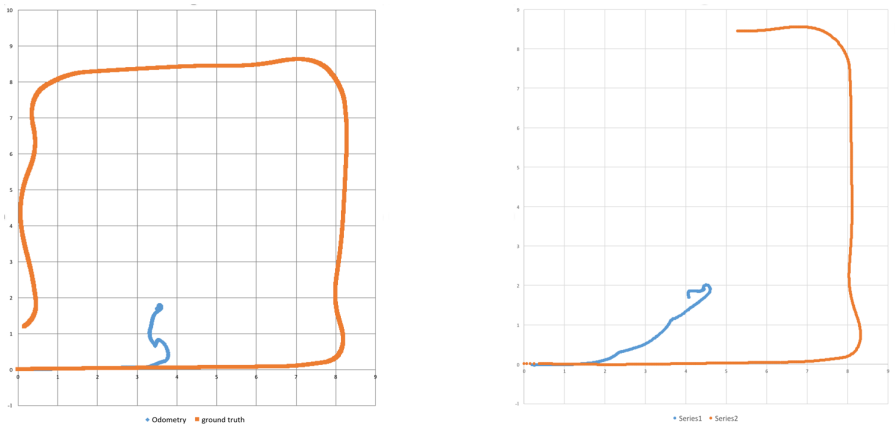

Figura 8: Comparação entre ground truth (laranja) e odometria (azul) para velocidade 0.4m/s.

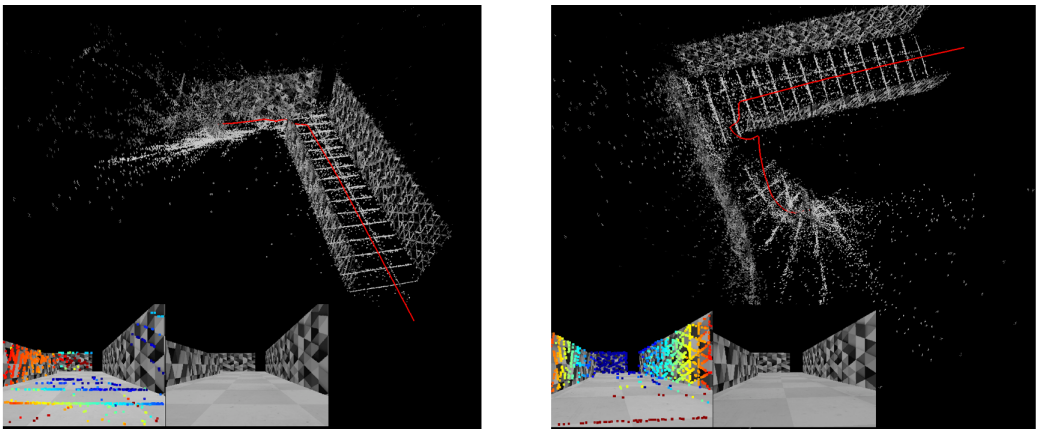

Figura 9: Observa-se que o algoritmo faz uma boa reconstrução do primeiro corredor, mas se perde ao analisar a curva

5 Conclusão

Em síntese, os resultados obtidos neste estudo vão de encontro aos apresentados na publicação original do DSO. De fato, o método se apresentou como uma opção atraente para uso em robôs por ser um método direto e esparso e por apresentar boa performance nos testes feitos pelos autores do algoritmo, mas a baixa performance obtida nos casos de teste deste trabalho reforçam a necessidade de maior investigação e avaliação das limitações do método em situações não cobertas pelos datasets usados originalmente.

Pôde-se concluir que a velocidade do robô parece não influenciar de forma significativa na performance do DSO, mas o tipo de textura captado pela câmera pareceu afetar a percepção do algoritmo. Vale a ressalva de que as condições em que foram realizados os testes deste estudo, como o uso de máquina virtual, podem ter afetado negativamente a performance, e portanto um estudo em condições mais adequadas seria uma evolução possível.

Por fim, por se tratar de uma publicação recente, não existem ainda avaliações formais que investiguem mais a fundo os resultados apresentados pelos autores do método e explorem possíveis limitações e fraquezas da abordagem; daí a relevância deste trabalho.

Referências

[1] D. Cremers, J. Engel and V. Koltun, *Direct Sparse Odometry,* In *arXiv:1607.02565*, October 2016.

[2] M. Quigley and K. Conley and B. P. Gerkey and J. Faust and T. Foote and J. Leibs and R. Wheeler and A. Y. Ng, *ROS: an open-source Robot Operating System,* In *ICRA Workshop on Open Source Software*, 2009.

[3] F. Fraundofera and D. Scaramuza, *Visual Odometry - Part1: The First 30 Years and Fundamentals*, In *IEEE ROBOTICS & AUTOMATION MAGAZINE*, pages 80-92. December 2011.

[4] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan and N. B. Ismail, *Review of visual odometry: types, approaches, challenges, and applications*, In *Springer Plus*. October 2016.

[5] D. Cremers, J. Engel and V. Usenko, *A Photometrically Calibrated Benchmark For Monocular Visual Odometry,* In *arXiv:1607.02555v2*, October 2016.

[6] K. Yousif, A. Bab-Hadiashar and R. Hoseinnezhad, *Intell Ind Syst* (2015) 1: 289.

[7] Textura de mosaico: `http://allswalls.com/images/colorful-shapes-wallpaper-8.jpg`. Acesso 12/2017.

[8] Textura de molduras: `http://resizing.info/openphoto.php?img=http://st.depositphotos.com/1252248/1705/v/950/depositphotos_17051369-White-empty-frame-hanging-on-the-wall.jpg`. Acesso 12/2017.

[9] DSO repository: `https://github.com/JakobEngel/dso`. Acesso 12/2017.

[10] V-Rep website: `http://www.coppeliarobotics.com`. Acesso em 12/2017.

[11] ROS website: `http://www.ros.org`. Acesso em 12/2017.

[12] Fabricante do Robotino: `http://www.festo-didactic.com`. Acesso em 12/2017.

[13] Página DSO: `https://vision.in.tum.de/research/vslam/dso?redirect=1`. Acesso 12/2017.