Localization and Mapping via 3d Deep Learning

Adam Rivelli, Ignacio Gavier, Victor Wong

Abstract

The localization and description of the environment of an autonomous agent is of broad interest, particularly for areas such as robotics and computer vision. Traditional Simultaneous Localization And Mapping (SLAM) techniques have been developed based on feature extraction using various statistical tools. Recently, deep learning modules have been incorporated to this problem due to their performance on computer vision tasks. However, traditional SLAM systems are still preferred since deep learning models do not achieve a complete understanding of a 3D environment. In this paper, we present a deep learning based approach to the SLAM problem, based on point cloud registration. The results show that this method is not yet able to reach the accuracy of existing methods, though further work could integrate our method as a module in a larger overarching SLAM system.

1 Introduction

Machine learning has had great advances in recent years on computer vision tasks. Computer vision has many applications in robotics, automatic navigation, augmented reality, among others. Some decades ago, these tasks were mainly performed with engineering techniques using statistical tools [9, 6, 10]. However, the incorporation of deep learning is becoming more common and produces results comparable to those hand-crafted [4, 5].

Despite advances in computer vision, there are still tasks that struggle to be solved purely with deep learning, such as Simultaneous Localization And Mapping (SLAM). SLAM is the problem of localizing an agent moving in a given environment while constructing a map of the environment. One of the major barriers that deep learning has in solving this problem is that current techniques do not allow reasoning about the semantics of a large 3D environment. Because of this, traditional SLAM methods, which use statistical techniques such as Kalman filters, Sequential Monte Carlo, or factor graphs, have outperformed deep learning methods up until now.

The mapping problem within SLAM can be addressed by storing the environment in 3D, or by projecting the environment onto a 2D plane, which is useful for planar settings. A recent method, MapNet [4] introduced a solution for the SLAM problem using deep learning, though they only build up a 2.5D map of the environment, *i.e.* a 2D map with features. In using a 2.5D map, a significant amount of information related to height is lost, so we propose to use a 3D map instead, and build up a global point cloud of the environment. In this work, we aim to implement a 3D SLAM method based on deep learning methods.

Section 2 provides an overview of related methods. Section 3 describes the method that we present in this paper. Section 4 provides and discusses the results we achieved. Section 5 concludes this paper.

2 State of the art and Related works

The problem of converting a sequence of RGB images into a camera map and localization is closely linked to robotics, so SLAM techniques have been implemented prior to the existence of deep learning. The traditional SLAM architectures that were developed have been polished into sophisticated modules, such as registration, tracking, mapping, closed-loop sensing, among others [7]. Due to the recent advances of deep learning in vision and memory tasks, SLAM architectures have been published where some of these modules use learnable parameters, resulting in models with performance comparable to the best SLAM techniques.

MapNet [4] is an architecture for solving the SLAM problem in which RGB-D (color and depth) videos are used to produce a 2D map with features. By maximizing the cross-correlation between the map obtained so far and the map of the current observation, they obtain the new position of the agent and update the map with the new observation using an LSTM. The robustness of this method is due to the fact that they perform cross-correlation at all locations on the map and in all directions, which allows the input video to have a low sample rate and still work. However, this feature also makes it inefficient, since the computational cost grows with $O(N^2)$ (or $O(N^3)$ if the mapping were 3D). In the Methodology, we propose a new architecture that allows to build a 3D mapping without maximizing the likelihood of the agent's position with cross-correlation, by using the technique known as Deep Global Registration (DGR).

DGR [1] is a method for finding the transformation between two overlapping but distinct point clouds. DGR is made up of a few key components: a feature extractor [2], a correspondence prediction mechanism, and a differentiable version of a classical registration algorithm [3]. Additionally, if DGR is not confident in its registration, it falls back on a classical point cloud registration algorithm, though we omit this in our application. DGR accepts two point clouds as input, and produces the transform between the two point clouds as output.

3 Methodology

Our method has two related goals: to reconstruct a complete scene from a series of point clouds captured by an agent, and to determine the agent's position at each point where a point cloud was captured. Our method centers around a registration module that finds the transformation that aligns two overlapping point clouds. Given a list of point clouds captured by the agent, we can use the registration module to compute a list of transformations between each consecutive pair of point clouds. Given these transformations, the first n transformations can be composed to transform the nth point cloud, which can then be added to the global map, a point cloud built up from the list of point clouds.

We also consider a variant where, instead of registering each point cloud only with respect to the previous point cloud, we register each point cloud with a map that has been built up incrementally from all previous point clouds. Also, we compare DGR's performance to that of RANSAC, a traditional registration algorithm. This algorithm is summarized in Algorithm 1.

Algorithm 1 Our SLAM algorithm.

procedure LOCALIZE(pointClouds)
  Input: pointClouds, a list of point clouds from an agent
  Output: map: a single point cloud built up from the input pointClouds
  trajectory: A list of points in $SE(3)$ indicating the agent's position when it recorded each point cloud
  $pos \leftarrow (0, 0, 0, 0, 0, 0)$ ▷ The origin where the agent starts
  $trajectory \leftarrow [pos]$
  $map \leftarrow head(pointClouds)$
  for each $cloud$ in tail($pointClouds$) **do**
    $T_{map->cur} \leftarrow$ Register(map, $cloud$) ▷ Any method to find how $cloud$ fits into map
    $pos \leftarrow pos$.transform($T_{map->cur}$)
    Append pos to $trajectory$
    $transformedCloud \leftarrow cloud$.transform($T_{map->cur}$)
    $map \leftarrow map$.merge($transformedCloud$)
  end forreturn (map, $trajectory$)
end procedure

4 Results

Dataset and implementation details. In order to test our model, we chose a dataset called LORIS [8], which is a set of video recordings captured by an agent moving in cafes, offices, corridors, and other indoor places. This dataset also measures the depth of each pixel using a D435i sensor, as well as the 6DoF for each frame using LIDAR technology. Although the sampling rate of this dataset is 30 fps, we down-sample it to 2 fps because the agent is moving slowly, and to decrease computational costs. In order to further reduce computational cost and memory usage, we align each point cloud with the previous point cloud produced by the agent, instead of aligning each new point cloud with the whole map.

From RGB-D to point clouds. The first step to be able to use the dataset is to preprocess the RGB-D image sequences by converting them into point clouds located in a reference frame where the origin is the current camera position. To do this, it is necessary to perform a change of coordinates from the image to the three-dimensional world using the intrinsic matrix K. In Figure 1, we can observe the mentioned transformation, where closer points are colored with blue, while farther points are colored with red.

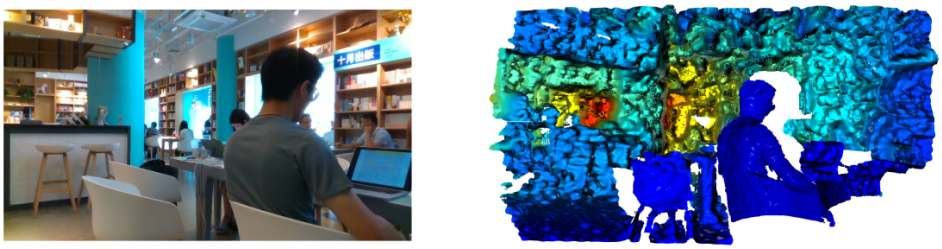

Figure 1: Conversion of RGB-D (we only show the RGB channels) image into a 3D pointcloud.

Global registration. For each step the camera takes, we perform the alignment between the two corresponding point clouds using the Deep Global Registration algorithm. In Figure 2, we

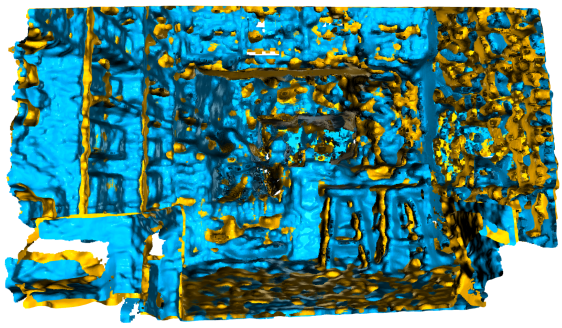

Figure 2: Two point clouds from consecutive positions of the agent being aligned using DGR.

show how pairs of point clouds seen from above, corresponding to consecutive camera positions, are aligned. However, we can notice that the alignment is not perfect: the yellow point cloud is located a little further than the cyan point cloud. We hypothesize that the differences between the dataset that DGR was pre-trained on - the 3DMatch dataset [11] - and the OpenLORIS dataset contributed to this error. As we can see in Figures 1 and 2, the OpenLORIS dataset produces noisy point clouds, especially compared to those present in the 3DMatch dataset.

Localization. The localization of the agent is computed through successive translation and rotation predictions using the registration algorithm. In Figure 3 we show the groundtruth trajectory of the agent and the predicted trajectory with DGR as the registration module. We also performed the test using another registration module, RANSAC, to corroborate both performances. It can be seen that both registration modules make good predictions at the beginning but then they quickly fall apart. We can also note that the algorithm using DGR is quite superior in predicting the agent's location compared to RANSAC, although the prediction is far from accurate compared to groundtruth. The quantitative values of MSError are indicated in Table 1. Most probably, the poor performance is due to the fact that the errors in the translation-rotation matrix predicted by DGR due to the noise of the aforementioned depths, accumulate for each step of the agent, causing a considerable loss of the final accuracy of the trajectory.

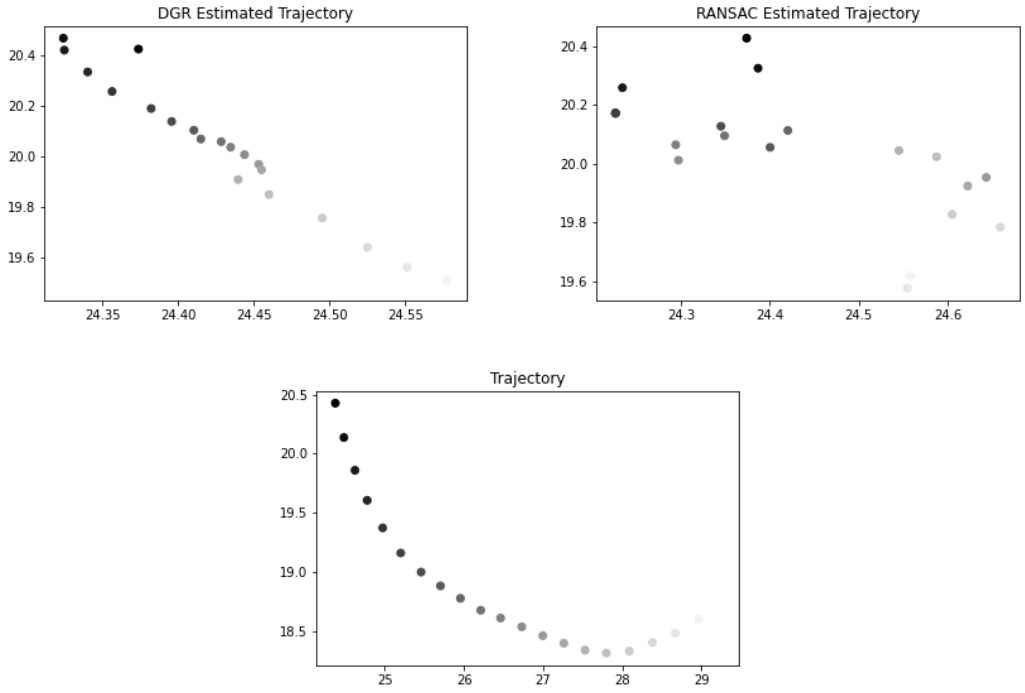

Figure 3: Trajectories of the agent predicted using DGR as a registration module (left) and RASNAC (right), and the groundtruth trajectory (bottom). The scale is in meters.

	MSError Angle	MSError Position $[m^2]$
DGR	**3.55**	**22.32**
RASNAC	4.04	22.95

Table 1: Table with MSError for the predicted trajectories for angles and positions.

5 Discussion

Compared to the current state-of-the-art, non deep learning related techniques, this method produced relatively inaccurate and inefficient results. Our method could be improved through various extensions to the work. One notable shortcoming of our method, is that DGR is trained on a different, much higher-resolution and cleaner dataset than we attempted to test on. Given our project's time constraints, we opted to not retrain DGR, though training on a dataset as challenging as OpenLORIS may allow DGR to perform better at this task. Additionally, our method only registered the current frame to the previous frame, rather than to the map that had been produced up until the current frame. Registering to the map as a whole could produce less noisy data, though it would further increase the computational cost of registration.

Overall, we believe that further work could improve upon this method, though we also believe that a robust SLAM architecture should use other data sources as well, such as odometry or visual features.

References and Notes

[1] C. Choy, W. Dong, and V. Koltun. Deep global registration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2514–2523, 2020.

[2] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[3] J. C. Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, Mar. 1975.

[4] J. F. Henriques and A. Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8476–8484, 2018.

[5] R. Li, S. Wang, and D. Gu. Deepslam: A robust monocular slam system with unsupervised deep learning. *IEEE Transactions on Industrial Electronics*, 68(4):3577–3587, 2020.

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[8] X. Shi, D. Li, P. Zhao, Q. Tian, Y. Tian, Q. Long, C. Zhu, J. Song, F. Qiao, L. Song, Y. Guo, Z. Wang, Y. Zhang, B. Qin, W. Yang, F. Wang, R. H. M. Chan, and Q. She. Are we ready for service robots? the OpenLORIS-Scene datasets for lifelong SLAM. In *2020 International Conference on Robotics and Automation (ICRA)*, pages 3139–3145, 2020.

[9] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[10] S. Thrun and M. Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[11] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, 2017.