Towards Visual SLAM with Memory Management for Large-scale Environments

Conference Paper · September 2017

CITATIONS
0

READS
51

4 authors, including:

Shaowu Yang
National University of Defense Technology
23 PUBLICATIONS **152** CITATIONS

SEE PROFILE

Towards Visual SLAM with Memory Management for Large-scale Environments

Fu Li[1,2⋆], Shaowu Yang[1,2 ⋆⋆], Xiaodong Yi[1,2], and Xuejun Yang[1,2]

[1] State Key Laboratery of High Performance Computing (HPCL), National University of Defense Technology.
[2] College of Computer, National University of Defense Technology, Changsha, China.

Abstract. Memory consumption of visual SLAM systems grows rapidly with their operation ranges increase. A well-designed organization scheme of map data is important for the scalability of visual SLAM in large-scale environments. In this paper, we present a novel visual SLAM system with an efficient memory management method to manage the map data using a spatial database. Experimental results on two popular public datasets demonstrate the efficiency and accuracy of our system.

Keywords: Visual SLAM, Memory Management, Spatial Database

1 Introduction

Simultaneous localization and mapping (SLAM) is originally proposed to locate a robot while simultaneously building a consistent map of an unknown environment. It has been a fundamental technology for autonomous navigation of robots in the past decades. Researchers have proposed different SLAM methods using different types of sensors, e.g. cameras or 2D/3D laser scanners. Specifically, visual SLAM systems utilize cameras to obtain sensor-data as input. Recently, with low cost cameras being widely used on robots, cell phones, and tablet devices, visual SLAM has become an attractive research focus in robotics, virtual reality (VR), and augmented reality (AR).

Some milestone visual SLAM systems have been proposed for constrained environments, such as MonoSLAM [1], PTAM [2], and ORB-SLAM2 [3]. These work mainly focuses on the accuracy and the efficiency of solving the SLAM problem using visual features, while keeping all map data in computer memory in cases that the SLAM process requires those data. As a result, how to conduct memory management of the map data, especially in large-scale environments, remains to be an open issue. In many real-world applications, e.g. large-scale or long-term operations of robots, it is a challenge to solve the memory management issue without affecting the accuracy and the efficiency of visual SLAM systems.

⋆ Project 91648204 supported by NSFC, project ZDYYJCYJ20140601 supported by NUDT, and project 201602-01 supported by HPCL.

⋆⋆ shaowu.yang@nudt.edu.cn

Large-scale environments challenge the scalability of visual SLAM systems. In most modern SLAM systems, the size of map grows unbounded with new places been explored, causing unbounded growth on memory consumption. Moreover, one trend of the SLAM research is to fuse data from multi-modal sensors to improve the accuracy and robustness of the SLAM in complex scenarios. Map data in such SLAM systems will cost much larger size of memory. Therefore, it is important to design SLAM systems with high efficiency on memory consumption. The focus of this paper lies in the memory management method of map data of visual SLAM to reduce memory consumption in large-scale environments.

Recently, research on visual SLAM considering memory consumption can be found in literature. The work in [4, 5] proposes to use a long-term memory (LTM) and short-term memory (STM) concept to solve dynamic environmental mapping. The work in [6–9] presents RTAB-Map for long-term large-scale operations of visual SLAM. RTAB-Map keeps the most recent and frequently observed locations in the working memory (WM), and transfers those locations which are less likely to be loop closures from WM to a SQLite database to reduce memory usage. However, the SQLite database in it limits its extension to really large-scale scenarios. The work in [10] presents C^2TAM based on a cloud computing framework. C^2TAM contains a cloud computing back end which publishes services for map data storage and map fusion of multi-robots, aiming at multi-robot cooperative tracking and mapping. This work focuses on mapping, map storage, map fusion, and real-time performance in the cloud end, while paying little attention on memory consumption in each single robot.

In this paper, we propose a novel visual SLAM system with memory management, to overcome two major challenges in reducing memory consumption of visual SLAM: efficient map data scheduling between the memory and the external storage, and map data persistence method (i.e., the data outlives the process that created it). We redesign the framework of a visual SLAM system to contain a SLAM front end and a database server end. The front end maintains a localization and a mapping process, as well as map data scheduling between the memory and the database. We propose a map data scheduling method to organize map data considering the local spatial character among keyframes. The server end provides data services from a spatial database for global map data storage and accessing. The performance of our system is demonstrated in the experimental results on TUM RGB-D [11] and KITTI [12] datasets.

2 System overview

The framework of our system is shown in Fig. 1, which includes the front end and the server end. We implement the front end based on ORB-SLAM2 by extending its tracking, local mapping, and loop closing module, and adding one novel memory managing module. The server end publishes data services for map data storage and access.

Tracking: The tracking thread processes each raw image frame for pose tracking, and decides which frame should be inserted into the map in the memory

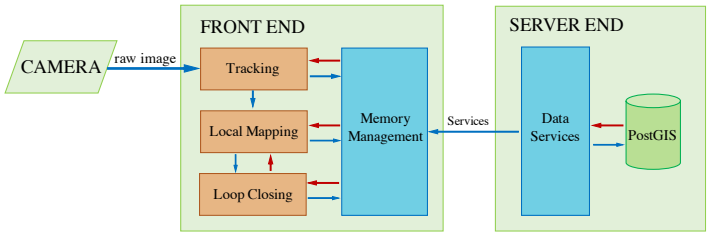


Fig. 1. The overview of our system.

as a keyframe. ORB features [17] in raw images are used to match to existing map points to achieve pose tracking. When a new keyframe should be added, it will be transferred to the memory managing module which schedules all map data.

Local Mapping: The local mapping module triangulates new map points from ORB features in keyframes. Moreover, it refines keyframe and map points in the local map, which is a sub-map of the global map affecting current pose tracking and mapping, by using local bundle adjustment (BA).

Loop Closing: The loop closing module detects loop closures among keyframes, and refines the global map using BA whenever a loop closure is found. All keyframes sharing similar ORB features with the new keyframe are used to compute similarity scores to the new keyframe. Several keyframes with the highest scores are selected as loop candidates. The candidates whose similarity transformations are supported by enough inliers will be accepted as loop closures.

Memory Management: The memory management module stores local map data, and schedules map data between the memory and the server end. When a new keyframe is inserted, it transfers related map data between the memory and the database server to update current local map. When a loop closure is detected, all pose data of keyframes and map points will be retrieved from the server to the memory for global map refinement by BA. When BA is finished, all optimized pose data will be updated to the database server to ensure data consistency.

Server: The server module manages map data in a database, and provides several data access services to other modules. It also provides variable interfaces which support complex service development for new application scenarios.

3 Memory management

As shown in Fig. 2, the memory management module mainly contains three parts, including the data scheduling, data driver, and the memory.

3.1 Data structure description

The core data structures in our system are keyframes and map points. Each keyframe mainly stores the camera pose, ORB features of the image frame, the

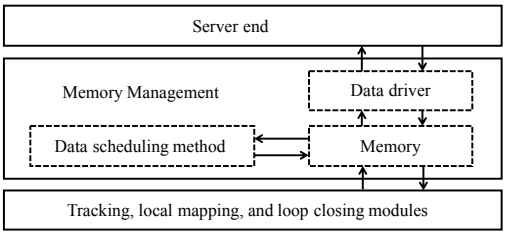


Fig. 2. The structure of memory management module.

identities (IDs) of all map points it observes and IDs of its co-related keyframes in the map. Each map point mainly stores its 3D position, its representative ORB feature, the IDs of keyframes in it is observed.

Another key conception is map which consists of keyframes and map points. The map elements in the map, i.e. keyframes and map points, make up the nodes, and the connections among the map elements make up the edges.

3.2 The data scheduling method

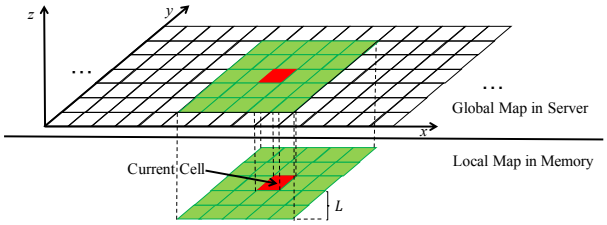


Fig. 3. The data-scheduling method conceptual graph.

The data scheduling method organizes the global map in the form of grids. It schedules the local map data to the memory and the rest of the global map data to the database. The local map data is selected as a subset of the global map data which affects the current tracking and mapping.

As illustrated in Fig. 3, the 3D global map is mapped into 2D grids by a horizontal decomposition. Then, it is organized in grid cells and managed in the database. Each grid cell has a unique ID, which is encoded as a 32-bit unsigned long integer to support large-scale maps. Multiple keyframes may be mapped to the same grid cell. Map points share the same cell ID as their source keyframe in which they are observed.The map data in a subset of grid cells from the server is selected and scheduled into the memory by the proposed data scheduling method with the following selection criteria: (1) the current cell in which the current working keyframe is located; (2) the cells within a distance L to the current cell. These criteria are designed for the reason that the similarities of visual features

in two keyframes reduce rapidly as the distance of the two keyframes increases, which will be further verified in Sect. 5.1.

3.3 Data driver

The data driver provides interfaces to transfer and retrieve data between the server end and the memory module. It packages the data and calls the services provided by the server to remotely transfer and retrieve the required data.

The possible data exchanges between the memory and server end modules can be listed as follows: (1) **Transferring local map data from memory to the server end.** The data driver accepts the map data from the memory, and packages them using the boost serialization library [13]. Then, it calls the server end service to save the data, and removes them from the memory. (2) **Retrieving local map data from the server end to the memory.** The data driver accesses map data from the server database according to the IDs of required keyframes and map points. Then, it deserializes the map data to corresponding data type, and inserts them to the local map. (3) **Retrieving geometric data from server end to the memory.** The data driver calls the service to retrieve geometric data, and deserializes the response data to keyframe poses and map point positions. This is done when geometric data is required by global BA. (4) **Updating geometric data from the memory to the server end.** The data driver packages all optimized pose or position data and calls the service to update the corresponding geometric data in the global map.

3.4 The memory module

The memory module stores the local map data and provides interfaces to access the keyframes and map points by their unique IDs.

To support mapping operations, map data in the memory must meet several constraints: (1) Local BA in the local mapping module requires keyframes that share map points with the current keyframe. (2) Loop closure detecting in the loop closing module requires keyframes that share ORB features with the current keyframe. (3) Global BA in the loop closing module requires all keyframe poses and map point positions.

The memory module schedules map data in real-time and runs in a separate thread. It monitors the current grid cell state, and uses the data scheduling interface to get the IDs of cells that need to be taken into or out of the memory. Then, it calls the interface from the data driver to transfer and retrieve relevant map data in cells. The above operations of the memory module can basically satisfy the constraints in (1) and (2). Besides, the memory module provides interfaces for adding keyframes or map points to the map in the memory, getting the connected data by the keyframe ID, getting the source keyframes of map points, and getting or updating geometric data in the global map from the server. The interfaces are designed to specifically meet the constraint in (3).

4 The server end

The server end module provides services for the front end to access the global map data. It consists of a model layer and a service layer.

The model layer supports the map data persistence and defines the basic data structures of map data in visual SLAM, mainly keyframes and map points. It stores and queries the map data from database tables, and provides basic operations on the data. We use ODB [14] to solve the issue of mapping a C++ object to a record in a database table, instead of using SQL. The ODB is an open source object-relational mapping (ORM) system, which facilitates the persistence of C++ objects to a relational database (RDBMS) without manual effort in processing tables, columns, or SQL. Furthermore, we use PostGIS [15] database to persist SLAM map data in large-scale operations. The PostGIS database is an extension to PostgreSQL to support spatial data analysis and processing.

The service layer implements the service logics on basis of the model layer. The services offered by the service layer are listed as follows: (1) Retrieving keyframes by cell ID; (2) Saving keyframes; (3) Retrieving map points by cell ID; (4) Saving map points; (5) Retrieving all geometric data; (6) Updating all keyframe poses and map piont positions.

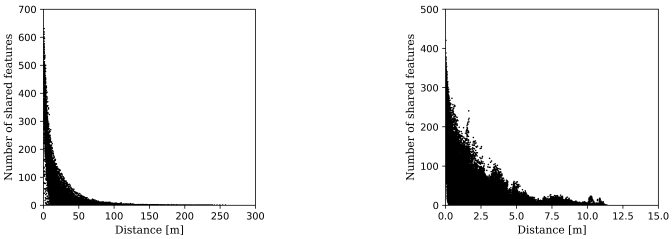
5 Experimental results

Our system is implemented in the Robot Operating System (ROS) [16], which provides a run-time environment that facilitates the communication between the front end and the server end. Experiments are conducted to evaluate the performance of our system compared with the original ORB-SLAM2 system. Two popular datasets, TUM RGB-D and KITTI dataset, are processed in the experiments. The KITTI dataset contains stereo sequences recorded from a car in urban environments, and the TUM RGB-D dataset contains indoor sequences from RGB-D cameras. The computer running the experiments features an Ubuntu 14.04 64-bit operating system, an Intel core i7-4790 (8 cores @ 3.6GHZ) CPU and 8GB RAM.

5.1 The data scheduling method verification

The data scheduling method in Sect. 3.2 is designed based on the visual similarities among keyframes. The similarity score is evaluated as the number of shared visual features between two keyframes. Keyframes with a similarity score to the current keyframe larger than $T_{threhold}$ should be kept in memory for local BA, and otherwise be moved to the database. In our system, $T_{threhold}$ is set to 10.

The similarity scores of any two co-related keyframes in our system for the two datasets are shown in Fig. 4. We can find that similarities among keyframes reduce as the distance of the two keyframes increases. Similarity scores among keyframes with distances larger than approximately 90 meters in KITTI sequence02 (in Fig. 4(a)) and larger than approximately 6 meters in TUM RGB-D Freiburg2 (in Fig. 4(b)), respectively, decrease to below $T_{threhold}$.

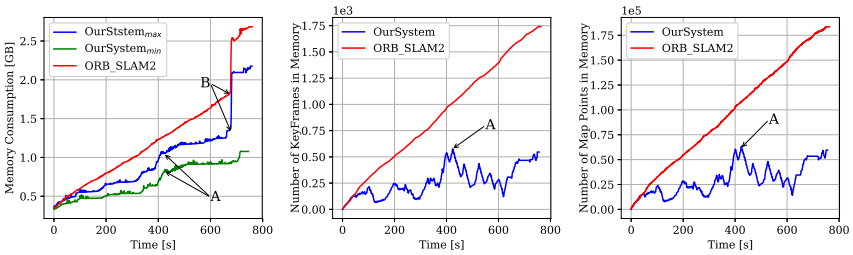


(a) (b)

Fig. 4. The number of shared visual features w.r.t. the distance of two keyframes.

5.2 Memory consumption

In this section, the dynamic memory consumption between our system and the ORB-SLAM2 system is compared. The dynamic sizes of maps of the two systems in the memory are also compared.



(a) (b) (c)

Fig. 5. Memory consumption of our system and ORB-SLAM2 on KITTI sequence02 dataset. (a) The memory consumption, (b) the number of keyframes, and (c) the number of map points.

Fig. 5(a) shows the memory consumption of both our system and ORB-SLAM2 on KITTI sequence02 dataset. Our system has much less memory consumption than the ORB-SLAM2 during the exploration. The memory consumption is mainly caused by map data in memory, i.e., keyframes and mappoints. As indicated by Fig. 5(b), 5(c), the map size in memory grows nearly linearly over time in ORB-SLAM2, while keeping below a threshold in our system. This is the major cause of the reduction of the memory consumption in our system. Memory consumption $OurSystem_{max}$ is reached when additional data are kept in the memory to fully support loop closing. $OurSystem_{min}$ is reached when only local map data is stored in the memory, without supporting loop closing. The memory consumption in our system theoretically lies between $OurSystem_{max}$ and $OurSystem_{min}$ when different loop closing strategies are implemented.

The memory consumption of our system appears a step (markded as A in Fig. 5(a)) when the size of the map reaches its maximum (marked as A in Fig 5(b)).

Due to the allocation mechanism in C++11, the memory consumption remains unchanged when the size of map reduces (after A in 5(b)). The free memory space is remained in the stack of a process to avoid applying for the system space too frequently. At about 700s, the memory consumption of both ORB-SLAM2 and our system appear surges. The reason is that the system needs to do the global BA when a loop closure is detected. Global BA expects the system to provide lots of free memory spaces for calculating the intermediate results.

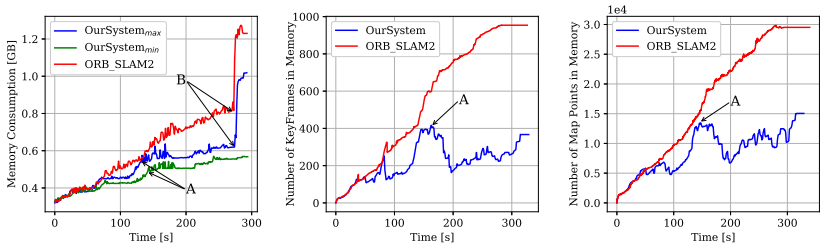


Fig. 6. Memory consumption of our system and ORB-SLAM2 on TUM-RGBD Freiburg2 dataset, with the same notations as in Fig. 5.

We have conducted a similar experiment on TUM-RGBD Freiburg2 with large loop dataset to further prove the advantages of our proposed system in memory consumption. The results are shown in Fig. 6, and we can draw similar conclusions with the case when using KITTI sequence02 dataset.

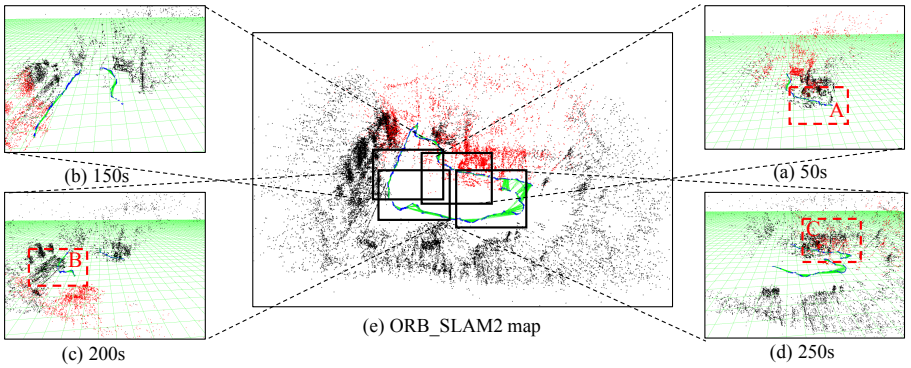
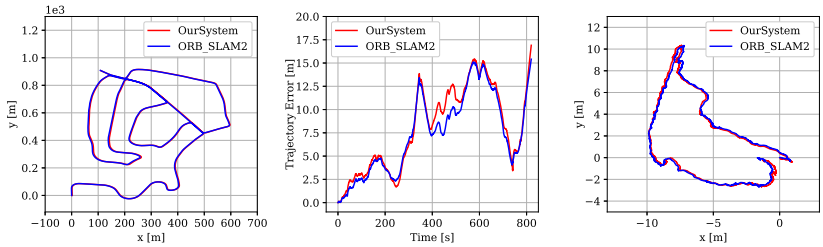


Fig. 7. (a),(b),(c),(d) Snapshots of the map in the memory of our system at 50s, 150s, 200s, 250s on the Freiburg2 dataset. (e) The snapshot of the map of ORB-SLAM2 when finishes running on the Freiburg2 dataset.

5.3 Map-data scheduling verification

Fig 7 shows the run-time snapshots of the map in memory at different time-stamps. As can be found in Fig 7(a), (b), some keyframes and map points at 50s (area A in Fig. 7(a)) are transferred to the server end and disappeared at 150s in Fig. 7(b), when the camera moves forward. As can be found in Fig. 7(d), some keyframes and map points in area C are retrieved from the server end, when the camera moves back. These results show qualitatively that our system can schedule the map data between the front end and the server end reasonably.

5.4 Trajectory comparison



(a) (b) (c)

Fig. 8. (a)Trajectories on KITTI sequence02, (b) trajectory error to KITTI sequence02 ground-truth, and (c) trajectories in Freiburg2 dataset of our system and ORB-SLAM2.

To demonstrate the accuracy of our proposed system, the camera trajectories of it running on KITTI and TUM RGB-D datasets are analyzed and compared with that of ORB-SLAM2. In Fig. 8(a), there is no obvious deviation between trajectories of our system and ORB-SLAM2 on KITTI dataset. Moreover, the trajectory errors to the ground-truth data in Fig. 8(b) shows that the two SLAM systems share similar tracking accuracy, which also indicates similar mapping accuracy. Similar results on TUM RGB-D dataset can be found in 8(c).

6 Conclusions

This paper proposes to reduce the memory consumption of visual SLAM in large-scale environments through an efficient memory management method. The resulting system contains a SLAM front end and a database server end. The experimental results demonstrate that our system efficiently reduces the memory consumption of visual SLAM, while maintaining its accuracy. The experiments are also shown in a video online[1]. The last contribution of this paper is that we provide the source code of our system to be publicly available [2].

[1] `http://v.youku.com/v_show/id_XMjc3MTU5ODU0OA`
[2] `https://github.com/lifunudt/M2SLAM`

References

1. Andrew J, Ian D, Nicholas D, and Olivier S. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6), 2007.
2. Georg K and David M. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.
3. Raul M and Juan D. Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras. *arXiv preprint arXiv:1610.06475*, 2016.
4. Feras D, Grzegorz C, and Tom D. Long-term experiments with an adaptive spherical view representation for navigation in changing environments. *Robotics and Autonomous Systems*, 59(5):285–295, 2011.
5. Feras D and Tom D. An adaptive appearance-based map for long-term topological localization of mobile robots. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3364–3369. IEEE, 2008.
6. Mathieu L and François M. Memory management for real-time appearance-based loop closure detection. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1271–1276. IEEE, 2011.
7. Mathieu L and Francois M. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.
8. Yunlong W, Bo Z, Xiaodong Y, and Yuhua T. Communication-motion planning for wireless relay-assisted multi-robot system. *IEEE Wireless Communications Letters*, 5(6):568–571, 2016.
9. Mathieu L and François M. Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2661–2666. IEEE, 2014.
10. Luis R, Javier C, and JMM M. C^2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.
11. Jürgen S, Nikolas E, Felix E, Wolfram B, and Daniel C. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.
12. Andreas G, Philip L, Christoph S, and Raquel U. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
13. Björn K. *Beyond the C++ standard library: an introduction to boost.* Pearson Education, 2005.
14. Sonia B, Claudio S, Domenico B, and Maurizio V. Odb-tools: A description logics based tool for schema validation and semantic query optimization in object oriented databases. *AI* IA 97: Advances in Artificial Intelligence*, pages 435–438, 1997.
15. Paul R et al. Postgis manual. *Refractions Research Inc*, 2005.
16. Morgan Q, Ken C, Brian G, Josh F, Tully F, Jeremy L, Rob W, and Andrew Y. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.
17. Ethan R, Vincent R, Kurt K, and Gary B. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.