A Cloud-Based Visual SLAM Framework for Low-Cost Agents

Jianhao Jiao(✉), Peng Yun, and Ming Liu

Robotics and Multi-perception Lab (RAM-LAB), Robotics Institute,
The Hong Kong University of Science and Technology, Hong Kong, China
jjiao@ust.hk

Abstract. Constrained by on-board resource, most of the low-cost robots could not autonomously navigate in unknown environments. In the latest years, cloud computing and storage has been developing rapidly, making it possible to offload parts of visual SLAM processing to a server. However, most of the cloud-based vSLAM frameworks are not suitable or fully tested for the applications of poor-equipped agents. In this paper, we describe an online localization service on a novel cloud-based framework, where the expensive map storage and global feature matching are provided as a service to agents. It enables a scenario that only sensor data collection is executed on agents, while the cloud aids the agents to localize and navigate. At the end, we evaluate the localization service quantitatively and qualitatively. The results indicate that the proposed cloud framework can fit the requirement of real-time applications.

Keywords: Cloud robotics · Mobile robot · Visual SLAM

1 Introduction

1.1 Motivation

Simultaneous Localization and Mapping (SLAM) jointly estimates the state of the robot and the map of environments. In the past few decades, SLAM, especially visual SLAM (vSLAM), has been an active research domain and many vSLAM algorithms have been presented to achieved great accuracy and robustness, such as [1–3]. However, vSLAM is both data intensive and computation intensive. In practical applications, the on-board resource is limited for service robots. For the applications such as augmented and virtual reality, it is hard to meet the computation and memory requirements.

This work was sponsored by the Research Grant Council of Hong Kong SAR Government, China, under project Nos. 16212815, 21202816 and National Natural Science Foundation of China Nos. 6140021318 and 61640305; Shenzhen Science, Technology and Innovation Comission (SZSTI) JCYJ20160428154842603 and JCYJ20160401100022706; partially supported by the HKUST Project IGN16EG12. All rewarded to Prof. Ming Liu.

M. Liu et al. (Eds.): ICVS 2017, LNCS 10528, pp. 471–484, 2017.
DOI: 10.1007/978-3-319-68345-4_42

The rapid development of network technology and the availability of commercial Internet servers make the solution to this dilemma envisaged. The cloud provides high-bandwidth connections, massive storage, data management, and computation. Regarding vSLAM, a cloud is able to process the steps such as local or global bundle adjustment, map fusion, and loop detection. Besides, the cloud can be understood as a center for knowledge sharing, giving robots the access to infrastructure, platform, software and data. For example, traditionally, each robot would have to explore and build its own map in large environments. But now, we use the cloud to save a live, global map of the large environment. If a new robot is introduced to the same environment, it will reuse the existing map without exploration effort.

There are few works with a lightweight and low-cost approach targeting complete SLAM applications. Therefore, it is worthwhile to investigate a proper solution for lightweight, low-cost robots navigation. To this end, we design a robot with a simple visual-inertial sensor, an ARM7 processor, and WiFi connection.

1.2 Contribution

In this paper, we build on the ORB-SLAM2 system (monocular part) [3], the place recognition work of DBoW2 [4] and *OpenResty*^{TM} platform[1], to design a cloud-based framework providing a localization service for low-cost robots. This service enables a robot with a smartphone-class processor to reuse maps and locate itself in environments. Compared with standard SLAM, it reduces much computation complexity. The contributions of this paper are:

1. We proposed a novel cloud-based framework which is able to provide different services and has two important features:
   (a) *Secure:* only authorized robots with right IP and password are allowed to connect to the cloud.
   (b) *Extensibility:* services are implemented with C/CPP and Lua scripts. Researchers are allowed to develop new applications freely.
2. We developed a lightweight localization service for the low-cost robots aided by the cloud. This service could reach real-time rate even though the bandwidth is limited.

1.3 Organization

This paper is organized into the following sections. Related work is described in Sect. 2. An overview of the system is introduced in Sect. 3, and details of our online localization service and framework are presented in Sect. 4, followed by the experiments shown in Sect. 5. Conclusion and future work are presented in Sect. 6.

[1] OpenResty: a registered trademark owned by OpenResty Inc. https://openresty.org.

2 Related Work

2.1 Low-Cost Localization Approaches

VSLAM is the primary approach for localization with low-cost sensors. An overview of SLAM was given by [5]. The most successful vSLAM systems currently in use are DSO [1] and ORB-SLAM [3,6]. DSO is a sparse and direct approach to monocular visual odometry, jointly optimizing the full likelihood for all involved model parameters to minimize the photometric error. OBB-SLAM is a feature-based monocular SLAM, which is able to close loops and reuse its map to achieve zero-drift localization in already mapped areas. However, ubiquitous optimization in these systems results in a high requirement for computational cost, which exceeds the capability of many mobile robots' processors.

Visible light communication (VLC) is a type of wireless communication technique and has many advantages such as low-cost and meeting the requirements of both illumination and communication. Researchers have tried to use VLC to deal with low-cost localization problem from a different perspective. In [7], Liu et al. discussed the feasibility of achieving accurate localization and preliminarily introduced a Gaussian Process to model the environmental light. By fusing the previous works [8,9], they demonstrated a low-cost VLC-based localization system in [10]. However, compared with vision version, VLC-based localization is only available in known places and limited by DoF. Fusing these techniques might overcome the limitations of each other [11].

2.2 Place Recognition Techniques

A survey by Williams et al. [12] compared the performance of appearance-based, map to map, and image to image methods for place recognition. Within appearance based approaches, a typical one is the FAB-MAP system [13]. It detects loops with an omnidirectional camera, obtaining great accuracy in long distance, but its robustness decreases when the images depict very similar structures for a long time. In contrast to the FAB-MAP, DBoW2 [4] uses bags of binary words obtained from BRIEF descriptors along with the efficient FAST feature detector to build a vocabulary tree offline. The work of Raulmur in [14] demonstrated DBoW2 to be very accurate and efficient.

2.3 Cloud-Based Framework

Based on Hadoop with ROS [18] as the master that manages all communications, DAvinCi was proposed in [19]. The goal of this architecture is to offload data- and computation-intensive tasks from the on-board resources on the robots to a backend cluster system. For proving its effectiveness, a parallel implementation on DAvinCi of Fast-SLAM was presented. However, the authors did not consider the data compression in any environment. They might face difficulties in transferring ROS message involving large data between the server and the robots.

The RoboEarth project [20] aims to develop a worldwide, open-source platform that allows any robot with a network connection to generate, share and reuse data. Later Rapyuta, an open source Platform-as-a-Service (PaaS) framework designed specifically for scalable robotic applications, was presented as the RoboEarth Cloud Engine in [21]. As a part of the RoboEarth project, it provides access to RoboEarth's knowledge repository, enabling the sharing of data and skills among robots. Based on Rapyuta, the author of [22] developed a parallel and dense visual odometry algorithm for collaborative 3D mapping on low-cost robots. Although many benefits of these frameworks are mentioned previously, there will be some potential drawbacks and challenges. In order to simplify the problem, most of the frameworks assumed that the online resource is unlimited. Actually, most resources in the cloud such as network bandwidth and CPU occupancy are limited.

Considering the real constraints, in [23], Riazuelo et al. presented the C2TAM framework for collaborative mapping in on multi-agents and discussed a solution about how to use the cloud's storage and computation resources properly. Furthermore, the RTAB-Map memory management approach [24] might be a good method to handle larger map's size in long-term and large-scale online mapping. In addition, fierce resource competition in multi-agents systems usually makes the bandwidth limited. Resource allocations strategies like [25] should be introduced to cope with this problem.

3 System Design

An overview of the cloud robotics system is illustrated in Fig. 1. The system consists of two components: a server (runs a cloud-based framework) and multiple robots. Generally, the simple processes like sensor data collection and data compression run on the robots. The computation- or memory-intensive tasks run on the server, which can concurrently handle a series of requests from different robots at the same time.

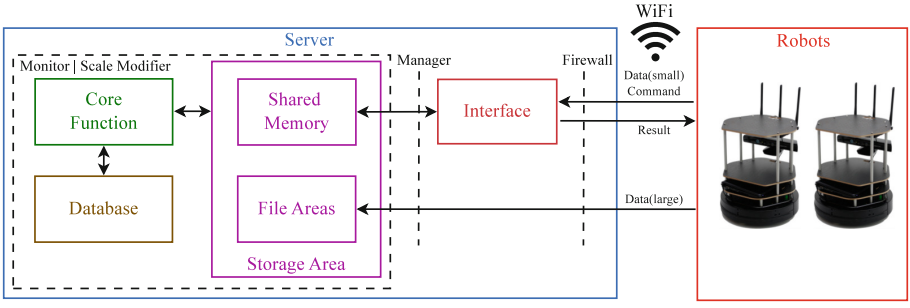

Fig. 1. An overview of the cloud robotics system

Considering hardware cost, real-time limitation, data I/O, network bandwidth, and computational requirements, a specific instance of our framework is described as following:

3.1 Robot (Client)

The low-cost robot is shown in Fig. 2. It equips with a single board computer (OrangePi Plus2[2]), a visual-inertial sensor($Loitor^{TM}$), and a WiFi module, uploading image features to the server to request its position. The details are shown in Fig. 3.

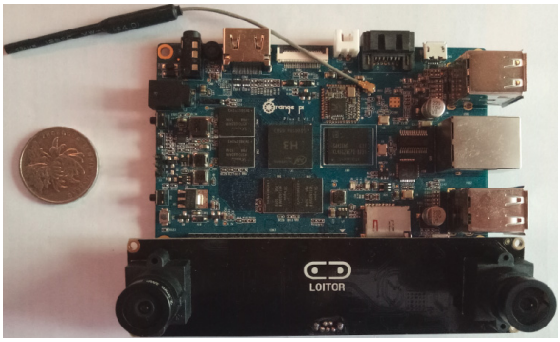

Fig. 2. The low-cost robot consists of a single board computer (<$40) OrangePi Plus2 and a visual-inertial sensor (<$90) $Loitor^{TM}$

OrangePi Plus2		Loitor	
CPU	Quad-core H3 Coretex-A7 processor	Camera	Global shutter | 24-65fps
Memory	2GB DDR3	Resolution	320x240 | 640x480 | 752x480
OS	Lubuntu	IMU	MPU-6050 | 200fps
Size	108mm x 67mm	Size	118mm x 30mm
Weight	83g		

Fig. 3. Details of OrangePi Plus2 and Loitor

3.2 Server

The server is a high-performance computer, running a cloud-based framework. It should have 5 main components: Interface, Shared Memory, File Areas, Core Function, and Database. If the size of some messages exceeds 100 Kbytes (called large messages), they will be uploaded from the robots through HTTP directly and then saved in the File Areas part. Other messages (called small messages) are sent through WebSocket by robots, decoded by the Interface part and then saved in the Shared Memory part.

[2] OrangePi Plus2: http://www.orangepi.org/orangepiplus2.

1. *Interface:* It sets WebSocket (small messages) and HTTP (large messages) as the communication protocol and decodes the small messages from String into JavaScript Object Notation (JSON)[3] format. It is also responsible for the connection to the robots.
2. *Shared Memory:* They are blocks of RAM that store messages (small) and can be accessed by the Interface part and Core Function part.
3. *File Areas:* They are areas in the hard disks that store image features posted from robots and can be accessed by Core Function part.
4. *Core Function:* They are mutually independent processes, providing localization services for robots. Their input is the image features and output is robots' position.
5. *Database:* It stores a feature database based on DBoW2 module and a trajectory database.

In order to ensure that the server could provide secure, stable and elastic compute services, we design the server with 4 additional parts: a firewall, a manager, a monitor and a scale modifier. The main function of each part is described below:

1. *Firewall:* It protects the server according to the secure rules, including open ports, access control lists and packet filter, etc.
2. *Manager:* It distributes data and tasks to Storage Areas and Core Functions according to different requests.
3. *Monitor:* It monitors and visualizes the server's states including CPUs and usage of the Storage Areas, the Core Functions and bandwidth, etc. Once some events get detected, the monitor will inform the scale modifier.
4. *Scale Modifier:* It scales the capacity of CPUs and memory up or down based on the online application's real-time demands.

After requests pass through the firewall, the manager distributes the requests to the proper Storage Areas and Core Functions. For example, it will distribute more resources to a complex request than normal ones. When the monitor detects errors, it will alter the scale modifier. Finally, the scale modifier will modify the scale of the abnormal parts.

4 Online Localization Service and Framework

In this section, we will describe the details of executing the localization service (Sects. 4.1 and 4.2) and building up the framework (Sect. 4.3).

[3] JSON: a lightweight data-interchange that is easy for humans to read and write. http://www.json.org.

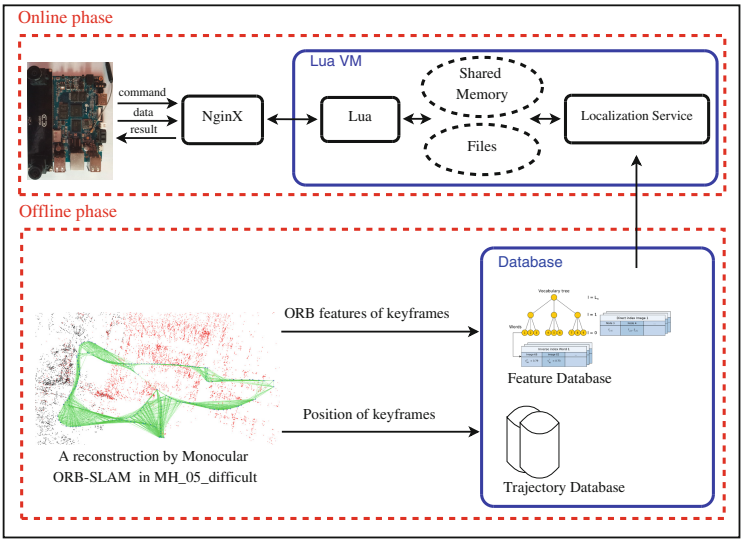

Fig. 4. Pipeline of the online localization service (top) and the off-line database creation (bottom)

4.1 Online Phase

Figure 4 introduces the pipeline of the online localization service and building process of the database. In the initialization step, the robot should firstly connect to the server. Once this connection is established, the robot starts capturing images and then uploading them to the server.

After receiving the new message, Lua virtual machine on the server will decode the command-type messages into JSON and save them in the Shared Memory.

The other part, called Core Function, runs simultaneously and provides localization service for the robot. Assuming that if some images captured by different cameras are matching with high scores, these cameras can be considered at the same place roughly. This assumption is used in the relocalization and loop detection step of some vSLAM systems. Each image is extracted ORB features and then searched its similarity in the feature database by using the bags of words place recognition module based on DBoW2 [4]. Finally, the robot's global localization will be retrieved in the trajectory databased and returned.

4.2 Offline Phase

Maps of large environments can be created by the Visual Monocular ORB-SLAM system. But different from [3], we only save the keyframes and disregard the 3D map points, co-visibility graphs, and essential graphs. The main reason is that after optimization, keyframes could reconstruct an environment in some extent. For each keyframe K_i, it stores:

1. The camera pose ${T_i}^w$, which is a rigid body transformation between the world to the camera coordinate system.
2. ORB features extracted in the keyframe.

The composition of the feature databases and trajectory databases are shown in Fig. 4. The visual vocabulary tree is created offline by discretizing all the descriptors of keyframes into K visual words. The feature database consists of vocabulary trees for different environments, direct indexes, and inverse indexes. The indexes are used for quick queries and feature comparisons. The trajectory database stores the pose and the number of each keyframe.

4.3 Framework

In this section, we will explain the main tasks and components of our framework. The framework runs on the server, providing configuration files, dependencies as well as independent environments for different robotic applications.

The framework is built on $OpenResty^{TM}$, which is a powerful web platform integrates Nginx core, Lua libraries, and LuaJIT. Researchers are allowed to implement web services on it with Lua scripts, Nginx C modules, and C/CPP programming languages. Compared with huge overhead of ROS, $OpenResty^{TM}$ is capable of handling 10K to 1000K connections on a single server.

Communication Protocol. The protocol defines the specification for message transmitting behaviors. Shown in Fig. 4, messages transmitted between robots and server could be classified into three types: command, data, and result. In every round, the robot sends command and data to the server and waits for the results. After establishing the connection, a Lua virtual machine is created by the server. The following computation tasks are completely processed in the Lua VM.

The command-type and result-type messages should be firstly encoded into JSON-type string format. JSON is a common communication format in web server, and many libraries available for JSON have been developed for various programming languages. We use the $RapidJSON$[4] to encode or decode the messages. But these JSON-type strings might be a burden to network bandwidth, so they will be further compressed before sending.

WebSocket is used to send command-type and result-type messages, which provides a full duplex, constant communication between robots and cloud in the framework. Compared with HTTP, WebSocket-based transmission helps save bandwidth resources hundreds of times under high concurrent connection. It also helps reduce 70% network delay compared with the HTTP long polling [26]. So it is efficient to send small messages through WebSocket.

Large messages will potentially consume plenty of bandwidth resources. Some results in [21] demonstrated that the delay and packet loss will increase if the

[4] RapidJSON: a fast JSON parser/generator for C++. http://rapidjson.org.

payload length of messages increases. At this point, the WebSocket protocol is not suitable for large messages transmission.

For instance, in the localization service or other complex services, robots need to offload some data like features or point clouds for storage or computation. These data might be larger than 100 Kbytes. Converting these data to JSON-type string format would result in an even larger message size because a float number might be transformed into multiple chars. As discussed above, the increasing pay length will limit the transmission of the messages. Therefore, we use HTTP to post files to the server instead of encoding them into String so that this limitation is avoided.

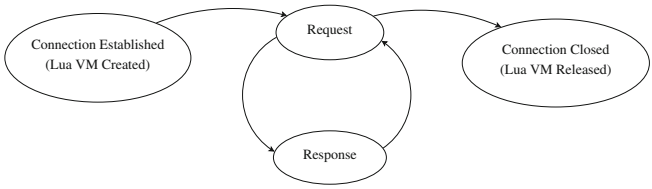

Fig. 5. The life of a Lua VM. When the connection is established, the Lua VM is created. The robotic service will be provided in a request-response way. Once the connection is closed, the Lua VM will be released

Isolated Computing Environment. When the connection between a robot and the server is established, a Lua VM is created to be used as an isolated environment. An isolated environment means that both memory and services are only provided for this robot and will not interfered with by other robots.

After creating a Lua VM, the service starts providing. In the next step, the robot will continuously send messages to the server and wait for the results. If the robot does not need services anymore (after sending a quit-type command), this connection will be closed and the Lua VM will be released simultaneously. The life length of Lua VM is as the same to the connection, which is shown in Fig. 5.

5 Experiment

In this section, we implemented the online localization service based on the cloud-based framework on the robot and ran the monocular ORB-SLAM system [3] on the same platform (called on-board ORB-SLAM). We compared the computational and memory cost of these two algorithms to demonstrate the feasibility and features of our proposed service.

5.1 Experiment Process

The cloud-based framework runs on a laptop (Intel Core i5, 2.5 GHz, 4 GB RAM). Both the laptop and the robot have a wireless connection to the campus Local Area Network. In this experiment, the server only handles a single request at the same time.

EuRoC dataset [27] contains 11 sequences recorded from a micro aerial vehicle (MAV), flying around two different rooms and an industrial environment. The sequences are classified as $easy, medium, difficult$ depending on MAV's speed, illumination, and scene texture. All the images were captured at 640×480.

In the offline phase, we ran the monocular ORB-SLAM in MH_01_easy to MH_05_difficult and then built up the databases with keyframes. Databases were saved on the server. In the online phase, we used HTTP to post new images to the server and used WebSocket to send the command-type and result-type messages (defined in Fig. 6).

There exist some feature matching errors because of the visual overlap between images. We firstly take 5 candidate images with a high score and then choose the most similar one.

What's more, we compared our localization service with the Monocular ORB-SLAM system [3]. We ran the ORB-SLAM system on the robot without connecting to the server and recorded the cost.

Command

```
{
  "type": "command"
  "id": "x"
  "content": "localization"
}
```

Result

```
{
  "type": "result"
  "id": "x"
  "content": "position(p_x, p_y, p_z, q_x, q_y, q_z, q_w)"
}
```

Fig. 6. Definitions of command-type and result-type messages in the localization service

5.2 Results

Cost and Bandwidth Analysis. Figure 7 shows the bandwidth required by the algorithm in MH_05_difficult. The red lines show the required bandwidth for the data from the client to the server, which are recorded every 0.5 s. Images are dominant in the data transmission so the command-type and result-type messages are not displayed. Note that the size of images determines the height of redlines.

The average bandwidth required was around 1.80 MB/s, which is less than the maximum available in a wireless connection (6 MB/s). The data transmission did not exceed the capacity of the current network.

Figure 8 shows a double-axis figure which compares the computational cost per frame of two algorithms (the localization service and the on-board ORB-SLAM) in MH_05_difficult. The average computational time of the on-board ORB-SLAM is around 332 ms/frame, while the time of our localization service

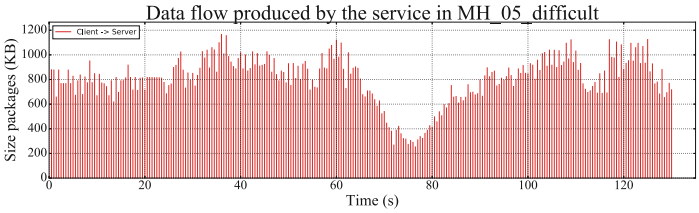

Fig. 7. Data flow produced by the localization service in MH_05_difficult of 2273 frames. Redline stands for images uploaded from client to the server. Compared with images, the command-type and result-type messages are small so they are not displayed. Each peak is recorded every 0.5 s. The average data flow for this localization service was 1.80 MB/s, below the usual wireless bandwidth which is 6 MB/s (Color figure online)

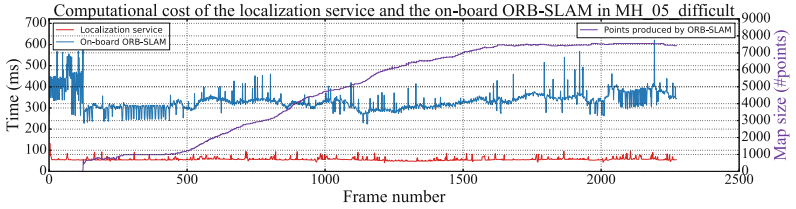

Fig. 8. A comparison of computational cost between the localization service and the on-board ORB-SLAM. The purple curve stands for the increasing map size caused by the ORB-SLAM, bringing memory burden to the robot (Color figure online)

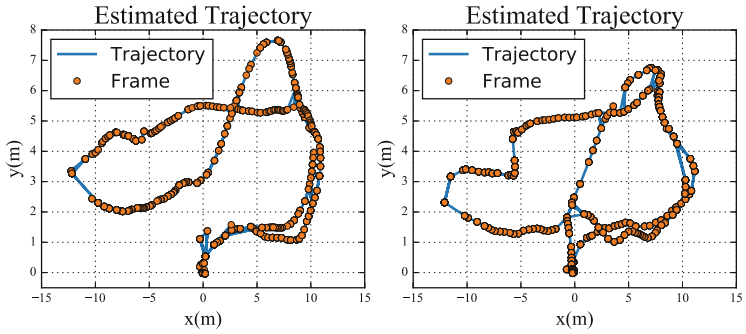

Fig. 9. Estimated trajectory in MH_04_difficult and MH_05_difficult. Note that the estimated trajectory in MH_04_difficult has less error matching

is around 57 ms/frame. Loading the vocabulary tree and saving the map are required by the ORB-SLAM, bringing extra memory and computational cost to the robot. Especially in long-term exploration, the increasing map size and optimization may exceed the robot's capability. So we consider that our proposed system is more suitable for low-cost robots.

Localization in the EuRoC Dataset. We choose MH_01_easy to MH_05_difficult to test our localization service. Figure 9 shows two examples of the estimated trajectory in MH_04_difficult and MH_05_difficult. We found that our proposed algorithm performed more accurate in MH_04_difficult than other sequences. Because the keyframes in MH_04_difficult have sparse distribution and less visual overlap, resulting in fewer error matchings.

6 Conclusion and Future Work

In this paper, we have presented a novel cloud-based framework that is built on WebSocket and *OpenResty*^{TM}, introduced its key components, and implemented an online localization service on it. Compared with other cloud robotics frameworks, it does not rely on ROS, send messages directly and is suitable for computation- and memory-intensive tasks. The localization service enables low-cost robots to relocalize themselves by reusing maps created by SLAM system. This service is able to handle each request within 60 ms. But influenced by the transmission delay, most of the time is used to wait for the new requests. So the transmission delay will be a bottleneck to our cloud-based application. Furthermore, based on the DBoW2 module, our localization service might cause some feature matching errors, which might limit the accuracy of our service. Finally, we ran a monocular ORB-SLAM on the robot (Cortex-A7 processor) and showed that our system is more suitable for low-cost robots.

The accuracy of the location service can be improved by using a dense- or semi-dense type SLAM system to reconstruct an environment, implementing an Inertial Measurement Unit (IMU) or a lightweight visual odometry on the robot to estimate its movement at a short time. In this system, the server and the robot are connected with WiFi, but WiFi is sometimes unstable and causes some transmission delay or mistakes. In addition, the framework is not able to handle requests from multiple robots at the same time. In the next step, we plan to take these factors into consideration to improve our system.

References

1. Engel, J., Koltun, V., Cremers, D.: Direct sparse odometry. IEEE Trans. Pattern Anal. Mach. Intell. (2017)
2. Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: large-scale direct monocular SLAM. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8690, pp. 834–849. Springer, Cham (2014). doi:10.1007/978-3-319-10605-2_54
3. Mur-Artal, R., Tardos, J.D.: ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. arXiv preprint arXiv:1610.06475 (2016)
4. Gálvez-López, D., Tardos, J.D.: Bags of binary words for fast place recognition in image sequences. IEEE Trans. Rob. **28**(5), 1188–1197 (2012)
5. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I.D., Leonard, J.J.: Simultaneous localization and mapping: present, future, and the robust-perception age. arXiv preprint arXiv:1606.05830 (2016)

6. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: ORB-SLAM: a versatile and accurate monocular slam system. IEEE Trans. Rob. **31**(5), 1147–1163 (2015)
7. Liu, M., Qiu, K., Che, F., Li, S., Hussain, B., Wu, L., Yue, C.P.: Towards indoor localization using visible light communication for consumer electronic devices. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), pp. 143–148. IEEE (2014)
8. Zhang, F., Qiu, K., Liu, M.: Asynchronous blind signal decomposition using tiny-length code for visible light communication-based indoor localization, pp. 2800–2805 (2015)
9. Qiu, K., Zhang, F., Liu, M.: Visible light communication-based indoor localization using gaussian process, pp. 3125–3130 (2015)
10. Qiu, K., Zhang, F., Liu, M.: Let the light guide us: VLC-based localization. IEEE Robot. Autom. Mag. **23**(4), 174–183 (2016)
11. Vadeny, D., Chen, M., Huang, E., Elgala, H.: VSLAM and VLC based localization
12. Williams, B., Cummins, M., Neira, J., Newman, P., Reid, I., Tardós, J.: A comparison of loop closing techniques in monocular SLAM. Robot. Auton. Syst. **57**(12), 1188–1197 (2009)
13. Cummins, M., Newman, P.: FAB-MAP: probabilistic localization and mapping in the space of appearance. Int. J. Robot. Res. **27**(6), 647–665 (2008)
14. Mur-Artal, R., Tardós, J.D.: Fast relocalisation and loop closing in keyframe-based SLAM. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 846–853. IEEE (2014)
15. Liu, M., Siegwart, R.: Topological mapping and scene recognition with lightweight color descriptors for an omnidirectional camera. IEEE Trans. Rob. **30**(2), 310–324 (2014)
16. Whelan, T., Kaess, M., Leonard, J.J., McDonald, J.: Deformation-based loop closure for large scale dense RGB-D SLAM. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 548–555. IEEE (2013)
17. Glover, A.J., Maddern, W.P., Milford, M.J., Wyeth, G.F.: FAB-MAP + RATSLAM: appearance-based SLAM for multiple times of day. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 3507–3512. IEEE (2010)
18. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, Kobe, vol. 3, p. 5 (2009)
19. Arumugam, R., Enti, V.R., Bingbing, L., Xiaojun, W., Baskaran, K., Kong, F.F., Senthil Kumar, A., Meng, K.D., Kit, G.W.: DAvinCi: a cloud computing framework for service robots. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 3084–3089. IEEE (2010)
20. Markus, W., Michael, B., Javier, C., d'Andrea, R., Elfring, J., Galvez-Lopez, D., Häussermann, K., Janssen, R., Montiel, J.M.M., Perzylo, A., et al.: RoboEarth. IEEE Robot. Autom. Mag. **18**(2), 69–82 (2011)
21. Hunziker, D., Gajamohan, M., Waibel, M., D'Andrea, R.: Rapyuta: the RoboEarth cloud engine. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), pp. 438–444. IEEE (2013)
22. Mohanarajah, G., Usenko, V., Singh, M., D'Andrea, R., Waibel, M.: Cloud-based collaborative 3D mapping in real-time with low-cost robots. IEEE Trans. Autom. Sci. Eng. **12**(2), 423–431 (2015)
23. Riazuelo, L., Civera, J., Montiel, J.M.M.: C2TAM: a first approach to a cloud framework for cooperative tracking and mapping

24. Labbe, M., Michaud, F.: Appearance-based loop closure detection for online large-scale and long-term operation. IEEE Trans. Robot. **29**(3), 734–745 (2013)
25. Wang, L., Liu, M., Meng, M.Q.H.: A hierarchical auction-based mechanism for real-time resource allocation in cloud robotic systems. IEEE Trans. Syst. Man Cybern. 1–12 (2016)
26. Lubbers, P., Albers, B., Smith, R., Salim, F.: Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. Apress, Berkely (2010)
27. Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M.W., Siegwart, R.: The EUROC micro aerial vehicle datasets. Int. J. Robot. Res. **35**(10), 1157–1163 (2016)