

MOARSLAM: Multiple Operator Augmented RSLAM

John G. Morrison, Dorian Gálvez-López and Gabe Sibley

Abstract To effectively act on the same physical space, robots must first communicate to share and fuse the map of the area in which they operate. For long-term online operation, the merging of maps from heterogeneous devices must be fast and allow for scalable growth in both the number of clients and the size of the map. This paper presents a system which allows multiple clients to share and merge maps built from a state-of-the-art relative SLAM system. Maps can also be augmented with virtual elements that are consistently shared by all the clients. The visual-inertial mapping framework which underlies this system is discussed, along with the server architecture and novel integrated multi-session loop closure system. We show quantitative results of the system. The map fusion benefits are demonstrated with an example augmented reality application.

1 Introduction

As robotic capabilities increase, cooperative robotic interaction is becoming more attractive. In order to operate in the same environment together, however, robots must simultaneously have an understanding of both the physical environment around them and their location within it. This requires a localization and mapping solution which can be shared between robots and permits relocalization within maps created by other devices. Also, to keep robots affordable, approaches based on low-cost sensors, such as cameras, are preferable to LIDAR-based solutions because their high cost can be prohibitive when developing distributed robotic applications.

John G. Morrison · Dorian Gálvez-López · Gabe Sibley
Department of Computer Science
University of Colorado
Boulder, CO 80309, e-mail: (john.morrison,Dorian.GalvezLopez,gsibley)
@colorado.edu

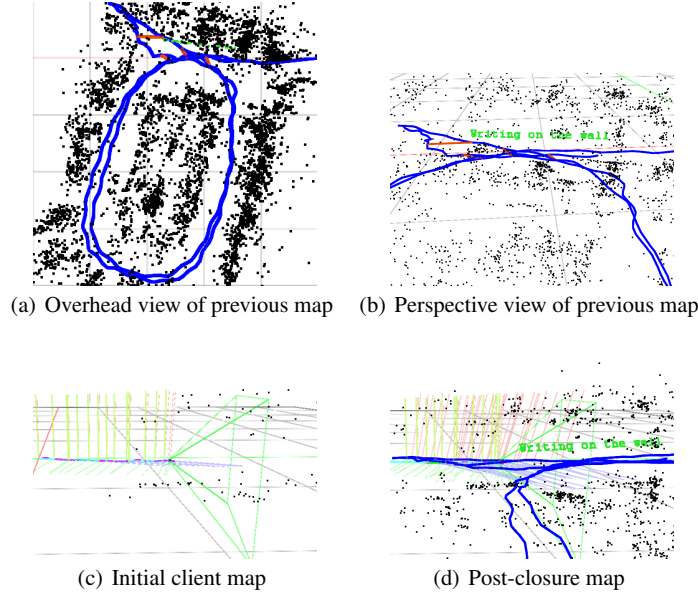


Fig. 1 Sample MOARSLAM operation. (a) A map previously created by a MOARSLAM client is stored in the server. The map and trajectory are depicted as blue edges, and the local loop closures as red edges. The landmarks are shown as black points and a virtual 3D object is also added, displaying the text “Writing on the wall”. (b) A zoomed in portion of the map. (c) Another client initiates a new mapping session with MOARSLAM in the same environment. (d) MOARSLAM soon recognizes a loop closure with the previously mapped trajectory and joins the two maps to provide the client with a richer map.

This paper presents a new system, MOARSLAM, for simultaneously building and sharing large-scale 3D maps from multiple devices equipped with either a monocular camera and an IMU or a stereo camera pair. It describes infrastructure, algorithms and core data structures for building and sharing arbitrarily scalable visual maps. The paper also presents a means for distributing physically tethered information, demonstrated here with an augmented reality example.

The presented system operates in a client-server architecture communicating over the network. This work defines a *client* as a camera- and network-equipped platform with computing abilities, such as a mobile phone or an autonomous robot. A *session* represents a single continuous MOARSLAM run by a client. A client starts a new session each time they launch MOARSLAM. The *server* in MOARSLAM is defined as a machine that stores a single representation of the accumulated global map. It offers an API interface to receive and distribute maps to provide endpoints for map related queries. The MOARSLAM client is capable of fully autonomous SLAM and only communicates with the server to share maps. The server’s simple and stateless API allows clients to easily integrate server queries into their processing when a network connection is available. The server API is also

purely image driven, removing the need for an accurate global location when interacting with the server. Fig. 1 shows an example of the capabilities of our approach.

The paper is organized as follows: Section 2 presents the related work of multi-device SLAM. Section 3 gives details about the map structure and its benefits. Section 4 gives an overview of the client-side SLAM processing pipeline which generates the initial session’s map. Section 5 describes the server component of MOARSLAM which is responsible for storing and joining multiple sessions’ maps. Section 6 shows the experimental validation of our proposal, and Section 7 includes a discussion on the potential limitations and future work for this approach. Section 8 concludes the paper.

2 Related Work

Multi-device SLAM has been approached in a decentralized fashion by several authors. Several vehicles on a sparse communication network create individual maps that fuse with their neighbors when it is possible, either by following a consensus policy [1, 21], or by distributing data association [12, 3]. Although decentralized systems can reduce the computation effort of each device, the complexity of their communications usually makes it expensive to obtain a global map. On the other hand, Forster et al. [7] and Riazuelo et al. [17] present a multi-threaded method for building maps using Micro Aerial Vehicles (MAVs) and mobile robots with a centralized system. Their approaches both seek to decouple the motion estimation and map building pipelines by communicating keyframe information to a central station where it is fused into a cohesive map structure. Loop closure events and map construction are handled on this central station while the client is left to perform visual odometry with only the map it is provided by the server. These solutions lower the computational cost on the client, but can reduce its autonomy in the face of intermittent networking. The system we propose is also based on a client-server architecture, but unlike approaches above, it allows clients to create individual maps even if connection to the server is lost. When connection is available, individual maps are fused by cross loop closures yielded by the server, as done by McDonald et al. [13]. However, unlike them, we do not require to extract new features to perform this operation since we exploit the features tracked by the front-end. In addition, we reuse the map graph to check for loop consistency.

Previous approaches for map representation fall largely into two camps: “hybrid” sub-mapping techniques and privileged frame fusion approaches. In fully fixed frame representations, such as the one by Özkucur and Akin [16], an expensive map merging algorithm is required to join measurements and objects in multiple maps together. These approaches offer simplicity in representation, but would not scale well because of the need to reprocess all data before merging maps. The sub-mapping techniques are based on limited-area metric maps, such as the occupancy grids used by Chang et al. [5] or the planar AR workspaces employed in PTAMM [4], the multiple map extension to PTAM [10]. These approaches apply a privileged frame

SLAM algorithm to populate their maps, but in order to bound complexity, after a map’s extent has grown too large they initialize a new and fully independent metric map. In PTAMM, they reuse their existing relocalization techniques to switch the system’s focus between maps, and Chang et al. [5] and McDonald et al. [13] connect maps using topological linkages based on odometry or “anchor nodes”. These approaches recognize the bounds of privileged frame representations, but still maintain a dependency on them. In their conclusion Castle et al. [4] discuss future work opportunities that this paper deals with, including IMU integration and the fusion of many sequences.

Current lines of research tend to augment feature maps with additional information, e.g. objects [20]. Furthermore, multiple robot or device localization allows clients to share these data in a consistent manner across maps. For example, the RoboEarth system [22] provides a framework to share semantic maps with object annotations among different robots. However, it does not define a mechanism to build a map cooperatively and requires clients to share an arbitrary reference frame. In contrast, our proposal deals with both cooperative mapping and map reuse, and makes it possible to consistently incorporate meta information relative to each client frame.

3 Scalable Mapping

This section describes the map structure which is the key to enabling multi-device map building, joining and updating in MOARSLAM. First the basic structure of the map is described, followed by a description of the particular attributes of the map structure which make multi-device mapping possible. Finally, the means through which external information is stored in the map is described.

3.1 Map Structure

The map is represented in the “continuous relative representation” of RSLAM [14]: an undirected graph of nodes, representing key frames taken at poses p , connected by transformation edges which encode their 6-DOF relative pose estimate, as seen in Fig. 2. Point landmarks l are stored as inverse-depth estimates back-projected (solid black lines) from the frame where they were first observed [6] back-projected from the pixel where they were first seen. Patches of size 9-by-9 pixels are stored for tracking the landmark. Each new observation of a landmark is stored with the frame from which it was observed (dashed edges). Additionally, the camera calibration, both the camera intrinsics and extrinsics, is stored for each session. This allows maps created in separate sessions to be used together. In MOARSLAM, a map is any set of frames which are connected through a set of edges, regardless of the session in which they were created.

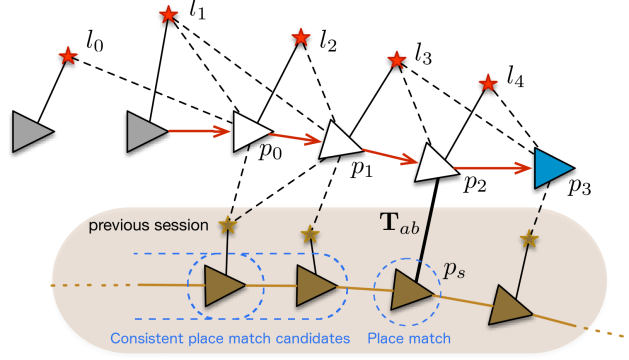


Fig. 2 Graphical representation of a map and of the constraints created after a loop closure. The poses p of a vehicle create graph nodes at every key frame. Nodes are joined by transformation edges and are associated to map landmarks l . When the loop detector retrieves a place match, after several consistent candidate place matches, a new edge T_{ab} is created to fuse maps between the different sessions a and b .

3.2 Multi-device Mapping

To enable global uniqueness of information created during each SLAM session, the components of the map require identification. Every time a client begins mapping, it creates a new *Session ID*, a UUID which is used to label every frame, edge, measurement, and landmark created during that session, along with an integer ID making each object unique within a session. The camera calibration used for map creation is also tagged with the session's ID. This identification provides uniqueness of sessions, frames, measurements, and landmark references during processing and transfer so the server and client can be confident that they are referring to the same data.

Using a single relative graph structure to represent a map allows for a consistent interaction between clients, whether information was created locally or fetched from the server. Queries on the map are performed as graph searches. Most searches are in memory and very fast, but the persistent map structure facilitates transparent loading from disk, as described below. This includes searching for visible landmarks, and estimating pose transformations between non-adjacent frames.

Unlike representations based on a single frame of reference, the relative graph structure of MOARSLAM allows sections of mapped environments to be referenced independently of other mapped locations. This means there is no need to share an entire graph or transform nodes before interacting with them. This ability to interact and independently alter sections of the map is important for running many operations in parallel, such as loop closures and multi-session mapping. Also, downloaded sections of map are available for use immediately because the system does not depend on knowledge of a global coordinate frame to make use of frames.

Thus, pieces of the global map may be downloaded as they are made available or as bandwidth allows.

As it is designed for large-scale mapping and long-term autonomy, MOARSLAM includes a persistent backend for its map structure. SQLite is used as an interface for persisting portions of the map which are not in use to disk. A simple SQL query of the unique identifiers described above allow frames and edges to be loaded from disk when they are needed. Such a backend structure works well with the sliding window filter optimization in MOARSLAM [14]. Frames are persisted in real time after they have exited the filter and are no longer needed.

3.3 Map Metadata

The relative map framework of MOARSLAM allows physical tagging of 3D space with external information. This information could be anything a robot or an end-user may want to know about a location. This metadata is stored along with a spatial transformation on the keyframe from which its location is visible. In this way, when sharing maps of an environment, all the labeled metadata is distributed with the physical description of the space. In Section 6, we demonstrate this capability by simple tagging of locations with augmented reality text, but more complex embedding is possible.

4 Client Processing

MOARSLAM operates in a client-server model. In contrast to Forster et al. [7] and Riazuelo et al. [17] where only a visual odometry system is run on the client, the MOARSLAM client runs a full SLAM pipeline and generates the relative map described in Section 3. The overall system architecture is outlined in Fig. 3. The client components are designed with full navigational autonomy in mind. This reduces reliance on potentially intermittent networking capabilities. Each client box in Fig. 3 is divided into several threads, as shown by Table 1.

The client’s main responsibility in this framework is to produce relative maps of its environment for sharing. These maps are produced online as the client processes images and IMU data. The client also periodically communicates with the server through the API described in Section 5 to query for global loop detections, upload and download maps.

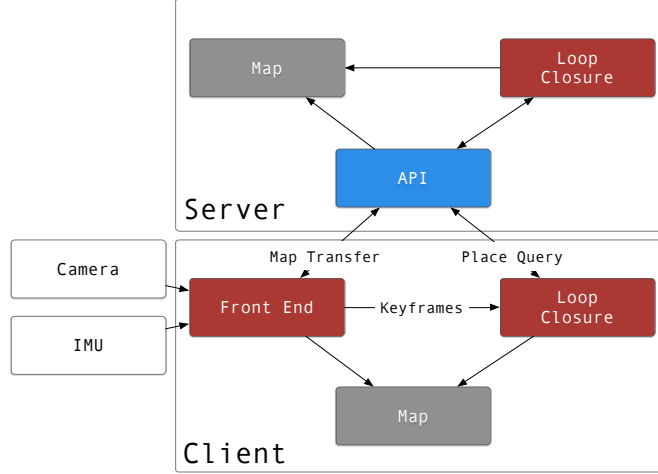


Fig. 3 System Architecture Outline.

4.1 Client Front-End

The client front-end is responsible for initial processing of all sensory inputs and produces a map. It leverages recent state of the art work on visual-inertial state estimation presented by Keivan et al. [9]. The front-end also asynchronously uploads all recent frames and edges (including their associated metadata) to the server.

The front-end is split into two threads which run asynchronously. The first thread, the tracking thread, tracks image point features using a patch-matching based scheme. This thread is designed to produce approximate pose estimates at a high rate. FAST corner features [18] are extracted from an image and compared with previously found map landmarks. This generates correspondences to estimate an essential matrix whose decomposition yields a relative pose. We disambiguate the translation scale by incorporating IMU measurements (or by triangulation with a stereo camera). This information is used in a Gauss-Newton optimization step to jointly improve the landmark and pose estimates. Using a set of heuristics based on inter-frame pose differences and the quality of feature tracking, some frames are selected as key frames. For every selected key frame, a local bundle adjustment is run to further improve estimates.

#	Algorithm	Purpose
1	Visual-inertial tracking and estimation [9]	Feature tracking and high-speed SLAM.
2	Adaptive-window bundle adjustment [9]	Error reduction in trajectory and landmark estimates.
3	Topological loop closure [8]	Previously visited place recognition.
4	Post-loop closure map relaxation	Drift reduction through pose graph relaxation.
5	Out-of-core SQLite map interface	Reducing memory by writing map data to disk.

Table 1 Threads and algorithms run by a MOARSLAM client.

The second thread in the front-end is an asynchronous adaptive-window bundle adjustment thread which grows or shrinks its window to maintain parameter observability and accuracy. This results in very high quality pose estimates at no cost to the overall front-end frame rate. This window can grow from the small size of the synchronous window up to hundreds of key frames, giving it the ability to capture motion and baselines not available to the fast-moving tracking thread.

4.2 Client Loop Closure

A third thread running on the client performs loop closures. For every key frame created during a mapping session, the loop closure system tries to detect places that have already been visited either by the current session or any other. This loop closure thread also queries the server to get matches with other previously uploaded maps. If a match is found, the client will download the server’s map of the surrounding area and integrate it into its own, joining the two disconnected maps.

The place matching is fully integrated in the system and takes advantage of all place information available to it, whether it was created during the current session or a previous one. MOARSLAM integrates the loop detector by Gálvez-López and Tardós [8] into its map graph and operates similarly. We store an image database to describe places as bags of binary words by using a single hierarchical vocabulary comprising 10^5 words, trained offline with millions of features obtained from independent data.

The loop closure thread computes ORB [19] descriptors around the corner features tracked by the front-end. This provides a small (~ 100) number of points which have been observed from multiple angles, are well distributed across the image and are associated to landmarks with estimated 3D poses. In addition, by reusing the front-end’s features the overhead of computing new features is avoided. These descriptors are converted into a bag-of-words vector and used to query the image database. The top-100 candidate matches are grouped together if their reference frames are close in the map, and the group with the highest aggregated similarity score is kept [8]. To avoid false detections, the candidates are accumulated until subsequent queries are resolved, as shown by the blue dashed circles in Fig. 2. When at least 3 of them are close each other, a loop closure attempt is made. By comparing ORB descriptors, 2D-to-3D correspondences between the current image and the matched 3D landmarks are found. Solving the perspective- n -problem yields a relative transformation that is later optimized by projecting map landmarks to find additional correspondences. If the transformation can be found, the place match is accepted and an edge is added to the map. This is illustrated by the thick edge created between poses p_2 and p_s in Fig. 2 that encodes the spatial transformation T_{ab} . Another thread performs asynchronous map relaxation after a successful loop detection to jointly improve all the pose estimations of the map.

5 Server

The MOARSLAM server component acts as an endpoint for storing, querying, and transmitting the maps which have been built by various MOARSLAM clients. The server provides a stateless API which allows clients to connect and disconnect without affecting the client-server interactions. Through the API, clients can perform three tasks: place recognition queries against previously uploaded maps, map uploading, and map downloading. The server uses the same map structure and place matching approach used on the client, simplifying implementation.

The communication between client and server is implemented using Node [2], an open-source C++ RPC and PubSub framework built on ZeroMQ and Google Protocol Buffers.

5.1 API

- **Place Recognition Query**

The client uploads an image and key frame information to the server to be matched against the server's database of places. The server's place recognition operates as described above in Section 4, but manages a larger joint map that comprises the places visited by each client. If a *place recognition query* request is satisfied, an edge is added between two different session maps, as shown in Fig. 2 or as seen in bright green in Fig. 6. A new edge that connects the query frame and the matched frame is returned to the client. Through this edge, the client can perform a *map download* request to access and include landmarks from the downloaded session and use them in its localization.

- **Map Download**

A client can request a section of graph by specifying a frame ID and a depth to which it would like a breadth-first search to be performed. This search will gather frames, edges, camera calibrations, and all associated metadata, and return it to the requesting client.

To contain the bandwidth usage of this operation, the maximum size of the returned map is fixed. In those cases that the limit is exceeded, the clients obtain "leaves" of the subgraph, which are the nodes at the edge of the fetched map. This gives the client locations to possibly ask for more map in a new *map download* request, if it is necessary.

- **Map Upload**

The client serializes frames, edges and associated places and send them to the server for later querying by other devices. These are inserted into the server's databases, which fuses them automatically if there are any conjoining edges.

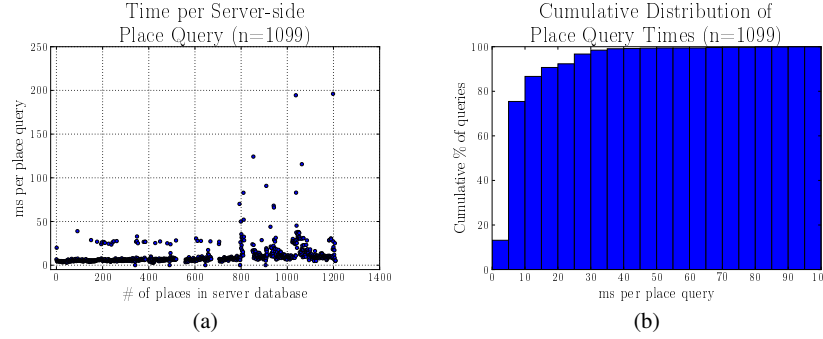


Fig. 4 Plots showing distribution of server-side place recognition query timings. Six data points are not shown on the plot because they fall far outside chart boundaries.

6 Experimental evaluation

6.1 Quantitative Results

To demonstrate an aspect of the scalability of MOARSLAM’s place query system, this section presents 1099 measurements of the run-time of the a single place query on the server. Each measurement begins after a request to the server has started processing and ends when a match is found or all potential matches have been eliminated. Fig. 4(a) plots the query processing time (in milliseconds) against the number of places in the database. The figure shows the low-constant linear growth rate for the majority of queries. Fig. 4(b) shows the heavy weighting of the measured times towards sub-10 millisecond responses. Both plots also demonstrate the long tail that can occur when interacting with a database, especially a simple on-disk storage interface like SQLite. The MOARSLAM server in this experiment was run on a 4-core 1.60 GHz Core i5 with a 5400 RPM HDD.

6.2 Qualitative Results

In order to qualitatively validate our multi-device SLAM system, we acquired three sequences of monocular images and inertial data with three Google’s Tango mobile phones. These describe different trajectories inside a room of approximately 45m². We then processed these sequences using a local server to create a global map that joins all the data.

We first ran our system with one of the sequences to create an initial map that was uploaded to the server. The two remaining sequences were run as two different client sessions interacting with the server through its API. Fig. 6 shows an example

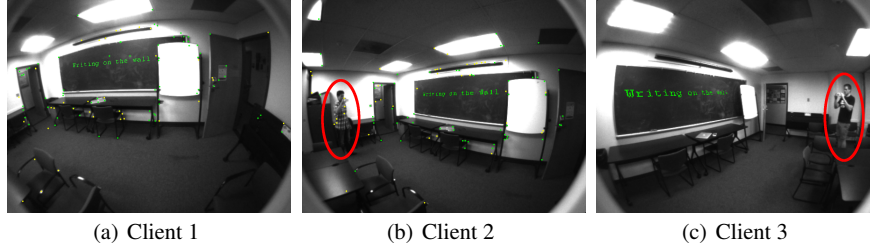


Fig. 5 Client 1 created a map with virtual text. Clients 2 and 3 simultaneously downloaded the map, including the virtual text. Clients 2 and 3 are circled in each other’s images for perspective.

of the map fusion process when one of the sessions finds a match with a frame on the server.

To illustrate the ability of our approach to augment maps and provide virtual objects to each client, we inserted a 3D object in the initial environment map displaying the text “Writing on the wall”, as it can be seen in Fig. 6. Fig. 5 shows a frame of each sequence after fusing maps, showing that the virtual object is correctly located for each client device.

7 Discussion

The relative framework is ideal for scalability and constant-time client operation, but it can present a challenge for users unaccustomed to considering non-euclidean representations. In particular, for *viewing* entire trajectories, it requires a global search and graph-relaxation to construct a consistent global view of the map. As this paper shows, global relaxation is not required for consistent metrically accurate multi-device augmented reality. Further, global relaxation is not needed for path-planning or obstacle avoidance [14]. In fact, a single global coordinate frame makes the estimation brittle, necessitating robust global loop closure algorithms [11, 15] and is only required for visualization. MOARSLAM mitigates the cost of global-optimization by caching the global map structure and only responding to map updates. It should be emphasized again that viewing global maps is not a requirement for localization or accurate metric interaction with the environment (e.g., for path planning, augmented-reality, obstacle avoidance, etc.). Similarly, while the relative manifold is accurate over short graph traversals of a few kilometers, error can accumulate around loops and cause a “tear” in the global visualization of a map. This is not a problem in the estimation as relative errors are close to optimal, it merely requires global-frame graph-relaxation to produce consistent *visualizations*.

Presently, MOARSLAM’s loop detection method is based on a single visual vocabulary. Previous research has shown that this approach is suitable for large heterogeneous environments mapped with very different cameras [8]. The next step is

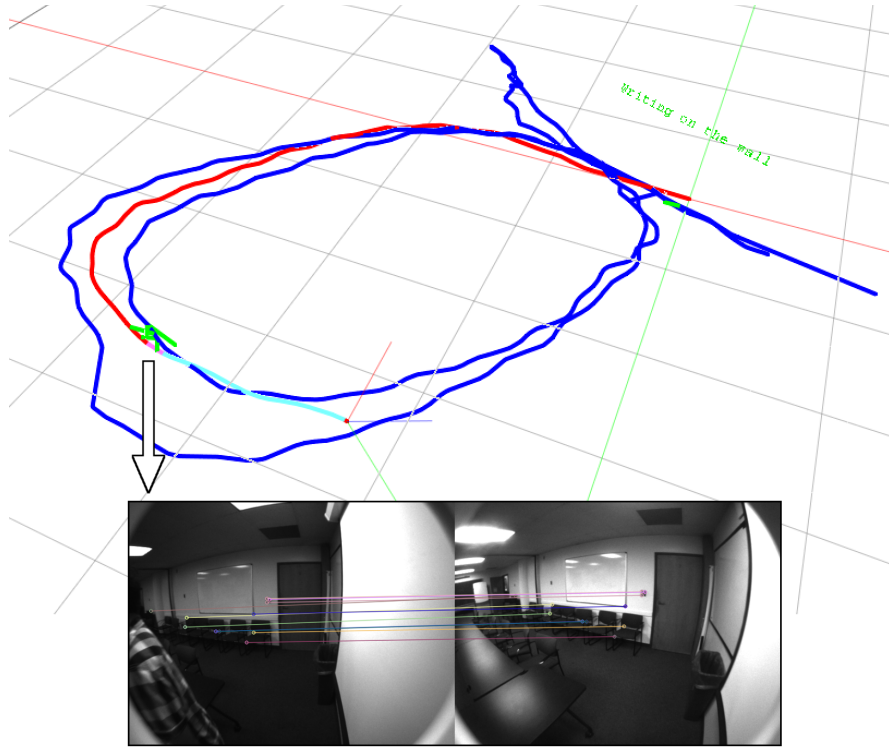


Fig. 6 Two maps are fused after a cross loop detection. The current image of the current session (red path) is matched with one the images of a previous session (blue path). An edge (green line) is added between the matching nodes, encoding a 6-DOF transformation computed from the correspondences between image features and map landmarks.

to research the limits of this approach when mapping trajectories of hundreds of kilometers overlapping in urban scenarios. In this context it is important to note that no algorithm is completely exonerated from false positives under all circumstances. Thus, long-term robustness can be achieved by applying a recent technique such as *Realizing, Reversing, and Recovering* (RRR) [11, 15], which accumulates several loop hypotheses to remove later those that are inconsistent. Note that the relative framework itself is not detrimentally harmed by false loop-closures, which can always be undone, reverting the map to it's prior condition without damaging the state estimate.

8 Conclusion

This paper presented MOARSLAM, a scalable, client-server-based system for multi-device SLAM. In addition, it showed the potential of this system for shared

augmented reality (AR) by distributing AR information as a component in the map through experiments demonstrating simultaneous AR from multiple perspectives in an indoor environment.

Sharing SLAM maps in a scalable manner is important for cooperative robotic tasks as robotic platforms become more capable. Sharing physically grounded information will allow teams of robots to operate together in an environment with confidence. MOARSLAM provides a foundation for sharing and reusing the ever more accurate maps created by modern SLAM systems.

Acknowledgments

This work is made possible with generous support from Google Project Tango.

References

- [1] R. Aragues, J. Cortes, and C. Sagues. Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics*, 28(4):840–854, Aug 2012.
- [2] ARPG. Node. URL <https://github.com/arp/Node>.
- [3] M. Bryson and S. Sukkarieh. Architectures for cooperative airborne simultaneous localisation and mapping. *Journal of Intelligent and Robotic Systems*, 55(4-5):267–297, 2009.
- [4] R. O. Castle, G. Klein, and D. W. Murray. Wide-area augmented reality using camera tracking and mapping in multiple regions. *Computer Vision and Image Understanding*, 115(6):854–867, June 2011.
- [5] H. J. Chang, C. S G Lee, Y. C. Hu, and Yung-Hsiang Lu. Multi-robot SLAM with topological/metric maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1467–1472, 2007.
- [6] J. Civera, Andrew J. Davison, and J.M.M. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, 2008.
- [7] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza. Collaborative monocular SLAM with multiple micro aerial vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3962–3970. IEEE, 2013.
- [8] D. Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.
- [9] N. Keivan, A. Patron-Perez, and G. Sibley. Adaptive Asynchronous Conditioning for Visual-Inertial SLAM. In *International Symposium on Experimental Robotics*, June 2014.

- [10] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.
- [11] Y. Latif, C. Cadena, and J. Neira. Robust loop closing over time for pose graph SLAM. *The International Journal of Robotics Research*, 32(14):1611–1626, 2013.
- [12] K. Y. K. Leung, T. D. Barfoot, and H. H. T. Liu. Distributed and decentralized cooperative simultaneous localization and mapping for dynamic and sparse robot networks. In *IEEE International Conference on Robotics and Automation*, pages 3841–3847, May 2011.
- [13] J. McDonald, M. Kaess, C. Cadena, J. Neira, and J. J. Leonard. Real-time 6-DOF multi-session visual SLAM over large-scale environments. *Robotics and Autonomous Systems*, 61(10):1144–1158, 2013.
- [14] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. RSLAM: a system for large-scale mapping in constant-time using stereo. *International Journal of Computer Vision*, 94(2):198–214, September 2011.
- [15] E. Olson and P. Agarwal. Inference on networks of mixtures for robust robot mapping. *International Journal of Robotics Research*, 32(7):826–840, July 2013.
- [16] N. Ergin Özkucur and H. Levent Akin. Cooperative multi-robot map merging using fast-SLAM. In *RoboCup 2009: Robot Soccer World Cup XIII*, number 5949 in Lecture Notes in Computer Science, pages 449–460. January 2010.
- [17] L. Riazuelo, J. Civera, and J. M. M. Montiel. C2TAM: a cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, April 2014.
- [18] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443. 2006.
- [19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011.
- [20] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *IEEE Computer Vision and Pattern Recognition*, June 2013.
- [21] R. Sharma, C. Taylor, D. W. Casbeer, and R. W. Beard. Distributed cooperative slam using an information consensus filter. In *AIAA Guidance Navigation and Control Conference*, pages 8334–8342, 2010.
- [22] M. Waibel, M. Beetz, J. Civera, R. D’Andrea, J. Elfring, D. Gálvez-López, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. Roboearth. *IEEE Robotics Automation Magazine*, 18(2):69–82, 2011.