Cloud-based Collaborative 3D Mapping in Real-Time with Low-Cost Robots

Gajamohan Mohanarajah, Vladyslav Usenko, Mayank Singh, Raffaello D'Andrea, and Markus Waibel

***Abstract*—This paper presents an architecture, protocol, and parallel algorithms for collaborative 3D mapping in the cloud with low-cost robots. The robots run a dense visual odometry algorithm on a smartphone-class processor. Key-frames from the visual odometry are sent to the cloud for parallel optimization and merging with maps produced by other robots. After optimization the cloud pushes the updated poses of the local key-frames back to the robots. All processes are managed by Rapyuta, a cloud robotics framework that runs in a commercial data center. The paper includes qualitative visualization of collaboratively built maps, as well as quantitative evaluation of localization accuracy, bandwidth usage, processing speeds, and map storage.**

***Note to Practitioners*—This paper presents an architecture for cloud-based collaborative 3D mapping with low-cost robots. The low-cost robots used in this work consist mainly of a mobile base, a smart phone class processor, an RGB-D sensor and a wireless interface. Each robot runs its own visual odometry algorithm, which estimates the pose of the robot using the color and the depth frames (images) from the RGB-D sensor. The dense visual odometry algorithm presented herein uses no image features and requires no specialized hardware. In addition to pose estimation, the visual odometry algorithm also produces key-frames, which is a subset of frames that in a way summarizes the motion of the robot. These key-frames are sent to the cloud for further optimization and merging with the key-frames produced by other robots. By sending only the key-frames (instead of all the frames produced by the sensor), bandwidth requirements are significantly reduced. Each robot is connected to the cloud infrastructure using a WebSocket-based bidirectional full duplex communication channel. The cloud infrastructure is provided using Rapyuta, a Platform-as-a-Service framework for building scalable cloud robotics applications. The key-frame pose optimization and the merging processes are parallelized in order to make them scalable. The updated key-frame poses are eventually sent back to the robot to improve its localization accuracy. In addition to describing the architecture and the design choices, the paper provides qualitative and quantitative evaluations of the integrated system.**

***Index Terms*—Cloud-based mapping, cloud robotics, Platform-as-a-Service, dense visual odometry**

Submission Type—Regular Paper

I. INTRODUCTION

The past decade has seen the first successful, large-scale use of mobile robots. However, a large proportion of these robots continue to either use simple control strategies (e.g. robot vacuum cleaners) or be remotely operated by humans (e.g. drones, telepresence robots). A primary reason for the lack of more complex algorithms in such systems is the cost (both direct and indirect) of onboard computation and storage.

The rapid progress of wireless technologies and the availability of commercial data centers, with high-bandwidth connections and highly scalable computation, storage, and communication infrastructures ('the cloud' [1]) may allow robots to overcome many of the current bottlenecks. Currently, several frameworks [2], [3], [4], [5] and robotic applications [6], [7] are being developed to exploit the cloud's potential for creating light, fast, and intelligent low-cost robots.

In this paper, we focus on using the cloud for mapping and localization – two of the most important tasks for any mobile robot. The process of simultaneously building a map and localizing a robot, also known as Simultaneous Localization and Mapping (SLAM), has been a research topic for many years and many SLAM algorithms have been proposed. Although the algorithms are increasing in precision, they require substantial onboard computation and often become infeasible when used for making larger maps over a long period of time. Furthermore, running everything locally also limits the potential for collaborative mapping.

A cloud-based parallel implementation of Fast-SLAM [8] was presented in [4] and showed a significant reduction in computation time. In this work, the authors presented a cloud infrastructure based on Hadoop [9] and received data from the robot using a common Robot Operating System (ROS) [10] master that managed all communications. Similar to [4], authors of [11] proposed a collaborative mapping framework where they moved the computationally intensive bundle adjustment process of the Parallel Tracking and Mapping (PTAM) [12] algorithm to a high performance server connected to the client computer. In addition to the above robotic scenarios, the Kinect@Home project [13] aims to develop a collection of RGB-D datasets through the use of crowdsourcing, by allowing any user with a Kinect and an appropriate web browser plugin to scan their environment. Once the dataset is uploaded, Kinect@Home performs a batch optimization and generates a 3D representation of the map for the user in the web browser.

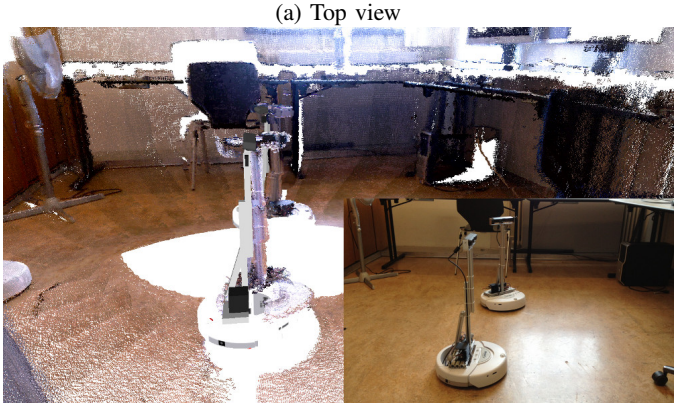
Matterport [14] is now developing a commercial system with custom cameras (similar to Kinect@Home), with the goal of making it easy for anyone to create 3D images of real-world spaces and share them online. Several centralized collaborative approaches that have the potential to run in a decentralized manner also exist. A 2D mapping system using manifold representation was introduced in [15], where the problem of

This work was done at the Institute for Dynamic Systems and Control (IDSC), ETH Zurich, Sonneggstrasse 3, 8092 Zurich. The contact author is Gajamohan Mohanarajah, e-mail: gajan@ethz.ch

–

Manuscript received month day, year; revised –.

(a) Top view



(b) Side view with a photo taken in a similar perspective.

Fig. 1: A point cloud map of a room at ETH Zurich built in real-time by the two robots shown in Fig. 3. The individual maps generated by the two robots are merged and optimized by processes running on a datacenter in Ireland. The robots are re-localized and the robot models are overlaid in the merged map.

map optimization and merging maps from different robots has been discussed. However, loop closure and map merging were only possible when another robot was recognized visually. In [16] the authors present a collaborative visual SLAM system for dynamic environments that is capable of tracking camera pose over time and deciding if some of the cameras observe the same scene; information is combined into groups that run the tracking together. More recently, several visual-inertial odometry systems [17], including Google's Project Tango [18] that runs on a custom cellphone with specialized hardware, has shown superior accuracy and consistency over the other approaches. But scalability, global optimization, and map merging remains open in the above mentioned visual-inertial systems.

This paper shows that low-cost robot platforms with a smartphone-class processor and a wireless connection are able to collaboratively map relatively large environments at quality levels comparable to the current SLAM methods. Furthermore, this paper shows a scalable approach to map optimization, storage, and merging of maps from different sources.

The main contributions of this paper are:

- Open source parallel implementation of dense visual odometry on a smartphone-class ARM multi-core CPU
- A novel cloud-based SLAM architecture and protocol, which significantly reduces the bandwidth usage
- Techniques for parallel map optimization and merging over multiple machines in a commercial data center
- An experimental demonstrator for quantitative and qualitative evaluation of the proposed methods

The remainder of this paper is organized as follows: We first give an overview of the system architecture in Sec. II. Onboard algorithms are presented in Sec. III. After presenting the data representation and communication protocol in Sec. IV we introduce the optimization and merging algorithms in Sec. V. Finally, the evaluation results of our implementation are presented in Sec. VI and we conclude in Sec. VII.

II. SYSTEM ARCHITECTURE

Real-time constrains, data I/O, network bandwidth, and computational requirements played an important role in the design choices of the proposed architecture. Generally, processes that were sensitive to network delays or which connected high-bandwidth sensors were run on the robot, while computation- or memory-intensive processes without hard realtime constraints were run in the cloud.

Our architecture, see Fig. 2, mainly consists of

- *mobile robot*: low-cost robots, each with an RGB-D sensor, smartphone-class processor and a wireless connection to the data center, see Fig. 3.
- *robot clone*: A set of processes for each robot connected to the cloud that manages key-frames and other data accumulation tasks, while updating the robot with optimized (or post-processed) maps. Currently, the *robot clone* sends the pre-programmed motion commands to the *robot*. This 'cloud-based control' functionality can be extended in the future to do motion planning based on the map being built, see Fig. 2.
- *database*: a database for storing maps. A relational (MySQL) database and a non-relational database (MongoDB) was used for comparison.
- *map optimizer*: Parallel optimization algorithm to find the optimal pose graph based on all accumulated key-frames. After each optimization cycle, the *map optimizer* updates the database and triggers the *robot clone* to update the robot with the new map.
- *map merger*: This process tries to match frames from different maps. Once a match is found, transformations between two maps are computed and the two maps are merged into a single map.

All computational processes run on Rapyuta [2], a cloud Robotic platform that manages the computational processes and handles the robots' bidirectional communication and authentication. See Sec. II-B for more details.

A. Robot

Our robots, shown in Fig. 3, consist mainly of off-the-shelf components. The differential drive base of the iRobot Create

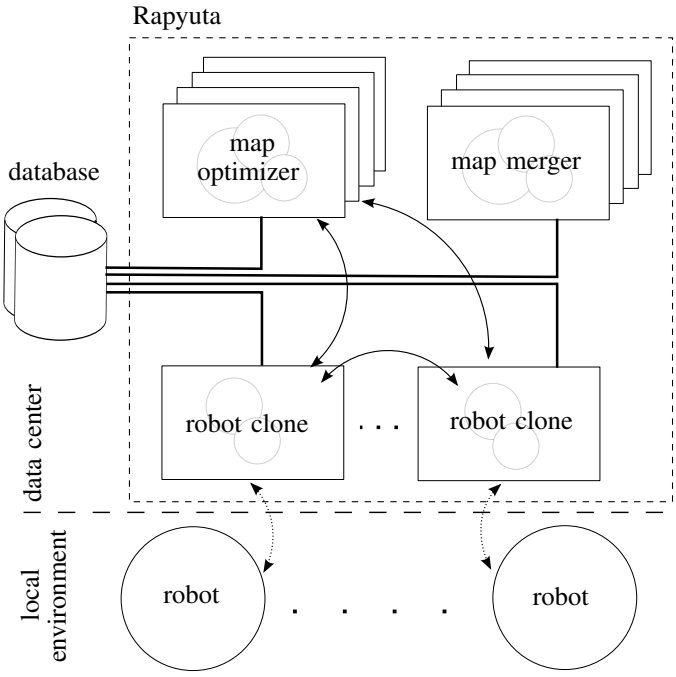


Fig. 2: The overview of the proposed architecture based on Rapyuta: Each robot has a corresponding clone in the cloud. The clone is a set of processes (light-gray circles) running under a secured computational environment (rectangular boxes). Every computational environment has its own ROS master and Rapyuta acts as a multi-master connecting processes running in different environments. Map optimization and merging are parallelized using multiple computational environments (stacked rectangles). All processes running inside the computational environments have a high bandwidth access to the database (cylinders). Robots have a WebSocket-based bidirectional full duplex connection (dotted curved lines) to the their corresponding clones in the cloud.

Fig. 3: The two low-cost (< 600\$) robots used in our evaluations: Each robot consists mainly of a differential drive base (iRobot Create), an RGB-D sensor (PrimeSense), an ARM-based single board computer (ODROID-U2), and a dual band USB wireless device.

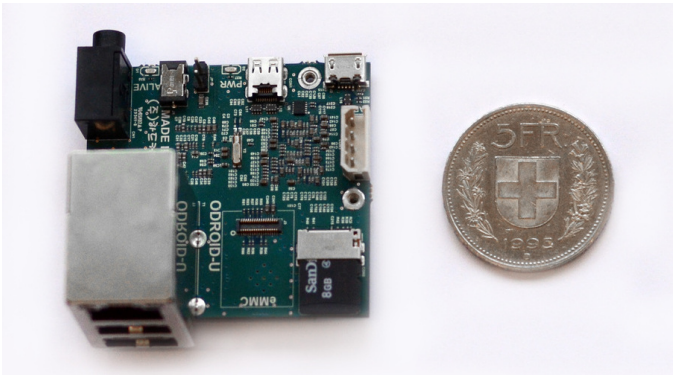


Fig. 4: The onboard processor Odroid-U2: 48×52 mm embedded board with a smartphone-class quad core ARM Cortex-A9 processor.

provides the serial interface for sending control commands and receiving sensor information. PrimeSense CARMIN 1.08 is used for the RGB-D sensing, and provides two registered depth and color images in VGA resolution at 30 frames per second. A 48×52 mm embedded board with a smartphone-class multi-core ARM processor is used for onboard computation. The embedded board runs a standard Linux operating system and connects to the cloud through a dual-band USB wireless device. In addition to running the RGB-D sensor driver and controlling the robot, the onboard processor also runs a dense visual odometry algorithm to estimate the current pose of the robot. The key-frames produced by the visual odometry are sent to the cloud processes through the wireless device. See Sec. III for more information on the visual odometry algorithm.

B. The Cloud and Software

We use Rapyuta [2], a cloud robotics platform we developed previously, to run all our processes in the cloud. Since Rapyuta uses the WebSocket protocol to communicate with the robots, the robots and mapping processes need not be in the same network as they were in [4] and [11]. This allows us to seamlessly connect our robots in Zurich, Switzerland to a commercial Amazon [19] data center in Ireland. Furthermore, since WebSockets allow for persistent connection between processes, the processes running in the cloud can push data/updates to the robots without the robots having to periodically poll for updates.

Rapyuta can spawn multiple secure ROS-compatible computing environments, launch processes inside these computing environments, and facilitate the communication between these processes (even across different computing environments). This allowed graceful scaling of *map optimizer* and *map merger* processes in experiments. Moreover, Rapyuta enables custom message converters to be employed between the robot and the cloud. This flexibility enabled us to use optimal compression schemes, resulting in a more than 50% reduction in bandwidth as compared to [11]. Visit http://rapyuta.org/ for

more details on Rapyuta.

III. ONBOARD VISUAL ODOMETRY

In order to build a map of the environment it is necessary to track the position of the robot over time. Although several methods (such as wheel odometry, visual odometry, and the use of Inertial Measurement Units) provide information on the relative motion of the robot, only a few of these (i.e. visual odometry) provide the option to remove the accumulated errors with global optimization. The dense visual odometry algorithm used on board the robots is largely inspired by [20], [21], and [22].

A. Preliminaries

This subsection defines some concepts and introduces the symbols used throughout the paper. Let

$$\begin{aligned}\mathcal{I} &:\mathbb{R}^2 \to [0,1]^3,\\ \mathcal{Z} &:\mathbb{R}^2 \to \mathbb{R}_+,\end{aligned}$$

represent the (color) intensity image and depth image of the camera respectively. To represent the camera's rigid body motion we use the twist vector $\xi \in \mathbb{R}^6$ and define $\hat{\xi}$ as

$$\hat{\xi} := \begin{pmatrix} 0 & -\xi(3) & \xi(2) & \xi(4) \\ \xi(3) & 0 & -\xi(1) & \xi(5) \\ -\xi(2) & \xi(1) & 0 & \xi(6) \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The over parametrized transformation matrix T can now be expressed as

$$T = \begin{pmatrix} R \in SO(3) & t \in \mathbb{R}^3 \\ 0^{3\times 1} & 1 \end{pmatrix} = \exp(\hat{\xi}).$$

Using a pinhole camera model, the projection π, and the inverse projection π^{-1} between the 3D point $\mathbf{p} := (X, Y, Z)$ and its corresponding pixel representation, $\mathbf{x} = (x, y)$ is given by

$$\begin{aligned}\mathbf{x} = \pi(p) &= \left(\frac{Xf_x}{Z} + o_x, \frac{Yf_y}{Z} + o_y\right),\\ \mathbf{p} = \pi^{-1}(\mathbf{x}, Z) &= \left(\frac{x - o_x}{f_x}Z, \frac{y - o_y}{f_y}Z, Z\right),\end{aligned}$$

where f_x, f_y denotes the focal lengths and o_x, o_y denotes the image center. Note that the second argument of inverse projection for our scenario comes from the corresponding depth pixel $\mathcal{Z}(\mathbf{x})$.

Given a frame, a tuple consisting of $\mathcal{I}$, $\mathcal{Z}$ and some other information (See Sec. IV), the warp of its pixel $\mathbf{x}$ to a frame with the relative pose M is given by

$$w(\mathbf{x}, M) := \pi\left(M\pi^{-1}(\mathbf{x}, \mathcal{Z}(\mathbf{x}))\right).$$

Finally, Key-frames are a subset of frames that in a way summarizes the full set. The key-frames are also used as a base/reference to represent the pose of other frames.

B. Dense Visual Odometry Algorithm

The dense visual odometry algorithm starts with an empty set of key-frames. When it receives the first pair of color and depth images, they are added to the map as an initial key-frame with the initial pose. A map in our scenario consists of key-frames and their corresponding poses.

After initialization, the dense visual odometry algorithm estimates the pose of the camera based on each incoming frame from the camera. This pose estimation is done by minimizing the photometric error between the intensity images of the current frame and the key-frame given by

$$\begin{aligned}R_k(M_k) &= \sum_{\mathbf{x}\in\Omega} (\mathcal{I}(\mathbf{x}) - \mathcal{I}_k(w(\mathbf{x}, M_k)))^2,\\ &=: \sum_{\mathbf{x}\in\Omega} r_{\mathbf{x}}^2(M_k),\end{aligned} \tag{1}$$

where Ω is a set of all pixels that are available in both frames that were not occluded while warped, and M_k is the relative pose of the key-frame with respect to the current frame. The key-frame that is closest to the last estimate of camera pose is used as the current key-frame.

To minimize the non-linear cost function given in (1) with respect to M_k we use the Gauss-Newton method for solving non-linear least-squares problems [23]. Here, the j^{th} iteration is given by

$$\begin{aligned}M_{j+1} &= \exp(\bar{\xi})M_j,\\ \bar{\xi} &= -(J_j^{\mathrm{T}}J_j)^{-1}J_j^{\mathrm{T}}r(M_j),\end{aligned}$$

where J_j is the Jacobian of the residual

$$r(M_j) := [r_{\mathbf{x}}(M_j)]_{\mathbf{x}\in\Omega}$$

and M_j is initialized with

$$M_0 = \begin{bmatrix} I_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}.$$

This iteration converges to

$$\arg\min_{M_k} R_k(M_k) = \lim_{j\to\infty} M_j.$$

At every iteration the Jacobian J_j can be calculated using the following chain rule

$$J_j = -\frac{\partial \mathcal{I}_k(w(\cdot))}{\partial w} \cdot \frac{\partial w(\cdot, \exp(\hat{\xi})M_j)}{\partial \exp(\hat{\xi})} \cdot \left.\frac{\partial \exp\hat{\xi}}{\partial \xi}\right|_{\xi=0}.$$

Note that the first term in the right-hand side is the color gradient and the other terms can be calculated analytically. The implementation of this algorithm was optimized to run on a multi-core ARM processor. All operations, such as color conversion, sub-sampling, image gradient computation, and 3D point re-projection are parallelized. These operations involve independent per pixel operations, so they can be easily parallelized by splitting all pixels between several CPU cores. To achieve this we use a Threading Building Blocks library [24], which provides templates for easy code parallelization. In particular the *parallel_for* and *parallel_reduce* templates are used heavily in our implementation. We also use the auto-vectorization tools of the GCC compiler, which automatically

replaces the regular instructions with specialized vector instructions where possible.

Since 3D point positions and image gradients are needed only for key-frames, they are computed only when a new key-frame is added (0-2 FPS depending on the robot speed). All images are stored in fixed-point values format (8-bit for intensity images and 16-bit for depth images), which may decrease the accuracy due to the rounding errors, but significantly improves the computational efficiency compared to processing images represented with floating-point values. With our implementation we were able to achieve a processing time of 15-20 [ms] for QVGA depth and color images. During this process, the visual odometry algorithm adds a new key-frame when the distance or the angle to the nearest key-frame in the map exceeds a predefined threshold.

IV. MAP REPRESENTATION AND COMMUNICATION PROTOCOL

Every map is a set of key-frames and a key-frame is a tuple represented as

$$(k, \mathcal{I}_k, \mathcal{Z}_k, q_k, t_k, \mathbb{I}_k),$$

where k is a global index of the key-frame, $\mathbb{I}_k$ is the intrinsic parameters of the camera, q_k the unit quaternion and t_k the translation vector. Note that q_k and t_k together represent the pose of the key-frame in the coordinate system of the current robot map. In the current implementation the global index k is a 64-bit integer, where the first 32-bits are used to identify the robot and the rest are used to index the key-frames collected by that robot. This indexing scheme saves approximately 4 billion key-frames from 4 billion robots, which is far beyond current needs.

The map is synchronized using the protocol shown in Fig. 5. When the visual odometry adds a new key-frame to the local map, it also sends one to the *robot clone*. All depth and color images are compressed with PNG for transmission. PNG is a lossless image compression that supports RGB and gray-scale images with up to 16-bit per pixel.

Once the key-frame has reached the *robot clone*, it is added to the database; the *map optimizer* process includes this key-frame in its next cycle. The *map optimizer* triggers the *robot clones* after the end of each cycle in order to update the local map on the robot. Once triggered, the *robot clone* gets the key-frame IDs of the local map on the robot, retrieves the updated key-frame pose from the database, and sends it back to the robot. The bandwidth requirement of this map update protocol is relatively low, since the update does not include any images/key-frame transmissions.

V. MAP OPTIMIZATION AND MERGING

The visual odometry that runs on the robot accumulates errors over time and causes a drift in the key-frame pose. This section presents the optimization techniques used to reduce the accumulated errors; these techniques work by minimizing error measures that include all acquired key-frames.

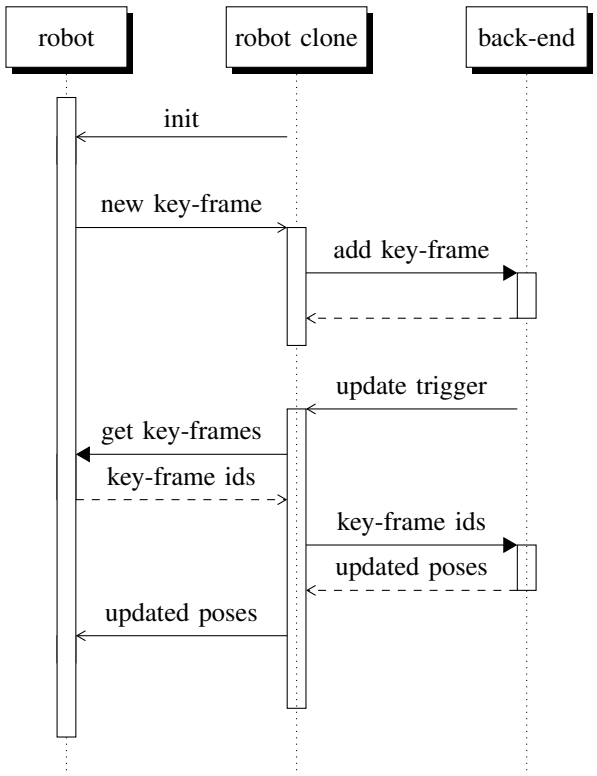


Fig. 5: Sequence diagram of the proposed map synchronization protocol: All key-frames from the robot are added to the back-end database. After every cycle of back-end optimization, the *robot clone* gets the local key-frame IDs from the *robot* and updates the local key-frame poses.

A. Map Initialization

Although this map initialization step is optional, it is recommended since it allows for the calibration of the camera intrinsic parameters. Further, where map initialization was used in experiments, the highly optimized initial map resulted in increased tracking stability.

During initialization the robot makes a 360 [°] in-place rotation. Assuming pure rotation allows to use well-established methods such as panorama optimization to be used. Our map-initialization is based on [25] and it globally optimizes all key-frame poses and the intrinsic camera parameters. When pure rotation is assumed, pixels from k'-th key-frame can be transformed to k-th key-frame by simply multiplying with the homography matrix

$$H_{kk'} = K R_k^{-1} R_{k'} K^{-1},$$

where R_k and $R_{k'}$ are rotation matrices of the key-frames k and k' with respect to a common reference, and K is an intrinsic calibration matrix parametrized by f_x, f_y, o_x, and o_y (see Sec. III-A). In order to find the optimal key-frame orientations and the intrinsic parameters, one must find the parameter vector

$$p = (f_x, f_y, o_x, o_y, R_0, ...R_N),$$

that minimizes the per-pixel error of each overlapping pair of

frames k, k' given by

$$E(p) = \sum_{k,k'} \sum_{\mathbf{x} \in \Omega_i} (\mathcal{I}_k(\mathbf{x}) - \mathcal{I}_{k'}(H(p)\mathbf{x}))^2$$

where $\mathcal{I}_k$ and $\mathcal{I}_{k'}$ are intensity images of the overlapping key-frames k and k'. The minimization of $E(p)$ with respect to p was performed using the Gauss-Newton method after parametrizing the updates using the Lie Algebra (see [25] for details). After the optimization, the floor of the map is selected using RANSAC and the XY-plane of the (world) coordinate frame was aligned with the floor.

B. Map Optimization

The global optimization of a map reduces errors accumulated during visual odometry, and consists of two main steps:

- Step 1: Construct a graph $\mathcal{G}$ where: 1) every key-frame of the map has a corresponding node; and 2) an edge between two nodes exists if the corresponding key-frames overlap and a relative transformation can be determined from the image data.
- Step 2: Solve the graph-based non-linear least squares problem given by:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{i,j \in \mathcal{G}} \| e(p_i, p_j, p_{ij}) \|_2^2,$$

where $\mathbf{p} = [p_i]_{i \in \mathcal{G}}$ is the pose vector of all key-frames, p_{ij} is the constraint due to the overlap of key-frames i and j (calculated in Step 1), and e is an appropriate error measure that describes how well the pair p_i, p_j satisfy the constraint p_{ij}. In our case we are using the error function that minimizes the translational error and the rotational error (magnitude of the real part of the unit quaternion that represents the rotational error) both equally weighted.

Once the graph is constructed, several state-of-the-art open source frameworks such as g2o [26] and Ceres [27] can be used to solve Step 2. Our architecture uses g2o for step 2. Since construction of the graph $\mathcal{G}$ in Step 1 involves the matching of key-frames, which is a computationally expensive task, we parallelize this process over multiple machines as shown in Fig. 6.

The *graph optimization* node retrieves pairs of key-frame indexes from the database, which don't have a transformation yet, and distributes these between *worker nodes*. Note that the *graph optimization* node only selects the key-frame pairs that are within a distance threshold in order to limit the exponential increase of the number of key-frame pairs.

The *worker nodes* try to compute the transformation between each pair of key-frames they receive. To compute the transformation, *worker node* loads the precomputed SURF keypoints for these key-frames and their respective 3D positions from the database and tries to find a transformation between them using RANSAC. If it succeeds, it saves the transformation to the database and proceeds to the next pair. Once all *worker nodes* have finished, the optimization node optimizes the error function, completing the optimization cycle. After every optimization cycle, key-frame poses are updated in the database and an update trigger is sent to the *robot clones* to update the local map on the robot. The graph structure of each map is stored as a table of pairs in a database and updated every time the new key-frames are added.

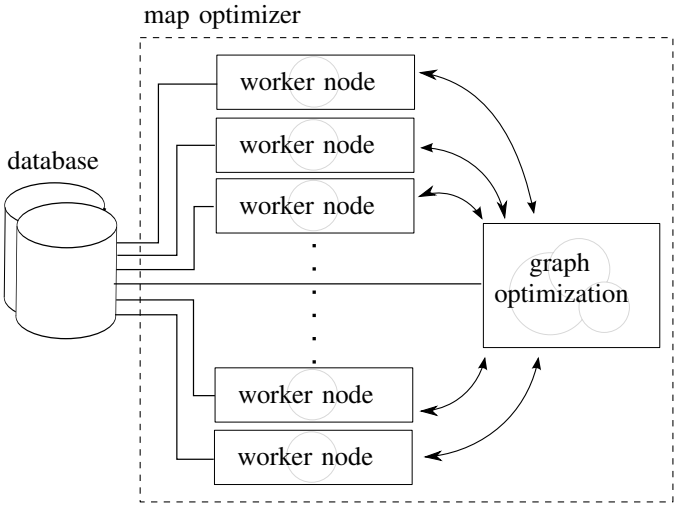


Fig. 6: Map optimization architecture: Pose graph construction is distributed among worker nodes and the constructed graph is optimized in the graph optimization node.

C. Map Merging

During collaborative mapping the robots can enter areas that have already been already explored by other robots. Being aware of the overlaps significantly decreases the mapping time and increases the map accuracy.

For the collaborative mapping, no prior knowledge on the initial robot poses is assumed and robots starts out with a separate map. The map merging runs as background process, continuously selecting a random key-frame from a map in the database and trying to find a matching key-frame from the other map. The process extracts SURF key-points from these key-frames and tries to match them using RANSAC. If a valid transformation is found, all key-frame poses of the smaller map are transformed into to the coordinate system of the other and the database entries are updated with the new values. Note that except for a minor difference in database update logic, the same *worker nodes* of the map optimization can be reused to parallelize map merging. Figure 1 shows a map merged from two robots and the re-localized robots in the new map.

VI. EVALUATION

The experimental setup for evaluation consisted of two low-cost robots (Fig 3) in Zurich, Switzerland and the cloud-based architecture (Fig. 2) running in Amazon's data center in Ireland. In addition to qualitatively evaluating the building and merging of maps created in different environments as shown in Figs 1 (72 key-frames) and 7 (423 key-frames), we quantitatively evaluated network usage, localization accuracy, and global map optimization times.

Figures 8 and 9 show the network usage of the robot executing a 360 [°] in-place rotation and a 2 [m] straight

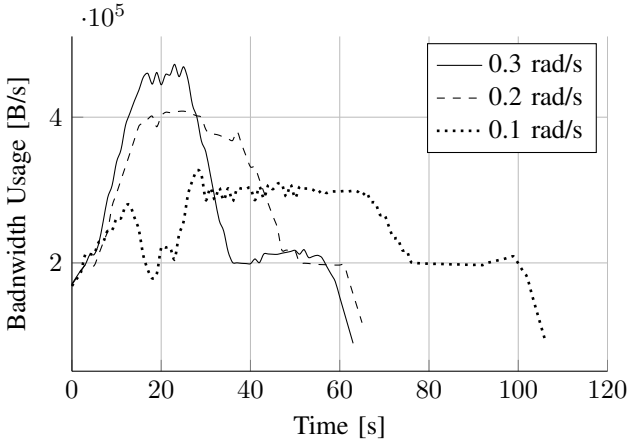


Fig. 8: Network usage in bytes per second for a single robot performing a 360 [°] in-place rotation.

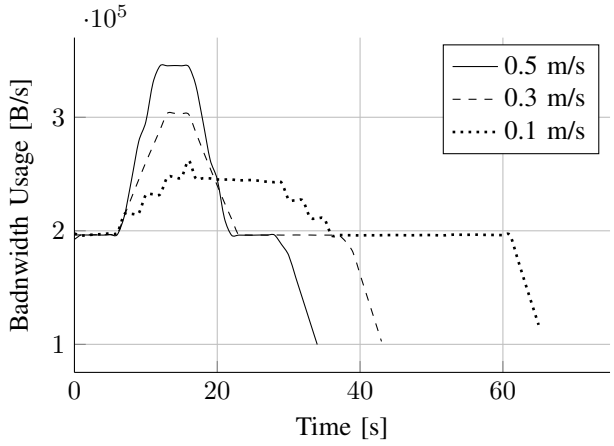


Fig. 9: Network usage in bytes per second for a single robot performing a 2 [m] straight motion.

motion with different speeds. It is clearly visible that bandwidth is proportional to the velocity of the robot, with the highest bandwidth about 500 [KB/s]. This value is half the bandwidth requirement of [11] (1 [MB/s]). For purposes of comparison, note that our cloud-based KinectFusion [28], a dense mapping algorithm, uses around 3.5 [MB/s] since all frames must be sent to the cloud for processing. For more details on this demonstrator and the video compressions used visit https://github.com/IDSCETHZurich/rapyuta-kinfu.

To evaluate the accuracy of visual odometry and influence of the global optimization, a high precision commercial motion tracking system was used. Figures 10 and 11 show the translation and rotation(yaw) errors error of the visual odometry with and without the cloud-based global optimization during a 360 [°] in-place rotation. Figure 12 shows translation error for a 2 [m] straight line motion. The yaw error during the straight line motion was below 0.01 [rad] for both the optimized and the non-optimized visual odometry. Note that, due to the relatively low visual features in the motion capture space, the maps of this space were of low quality compared to the ones given in Figs 1 and 7.

Finally, Fig. 13 shows the time taken for map optimization against the number of *worker nodes*. Although the processing time initially decreases with the number of *worker nodes*, this decrease later vanishes due to communication latencies. The measurements also show that the gain due to parallelization is significantly more for larger sets of key-frames. To reduce latencies due to database access during map optimization, we compared a relational and a non-relational database with respect to their I/O speeds. MySQL was used to represent relational databases, whereas MongoDB was used to represent non-relational databases and the results are shown in Figs. 13a and 13b. Although both databases gave a similar performance with respect to speed, using the `JOIN` clause of MySQL (join clause combines records from two or more tables in a database), a significant amount of computation was offloaded from the *graph optimization* node to the database during the key-frame pair selection (see Sec. V-B).

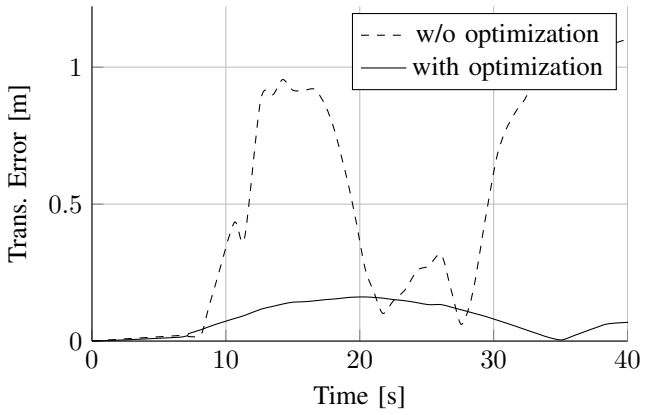


Fig. 10: Translation error of key-frames extracted by visual odometry during a 360 [°] in-place rotation with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

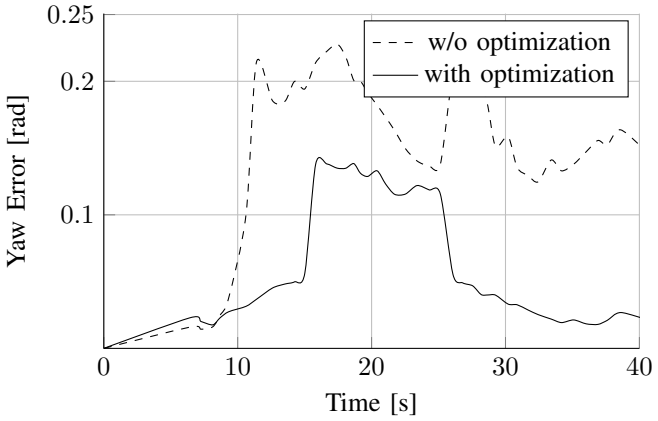


Fig. 11: Rotation error of key-frames extracted by visual odometry during 360 [°] in-place rotation with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

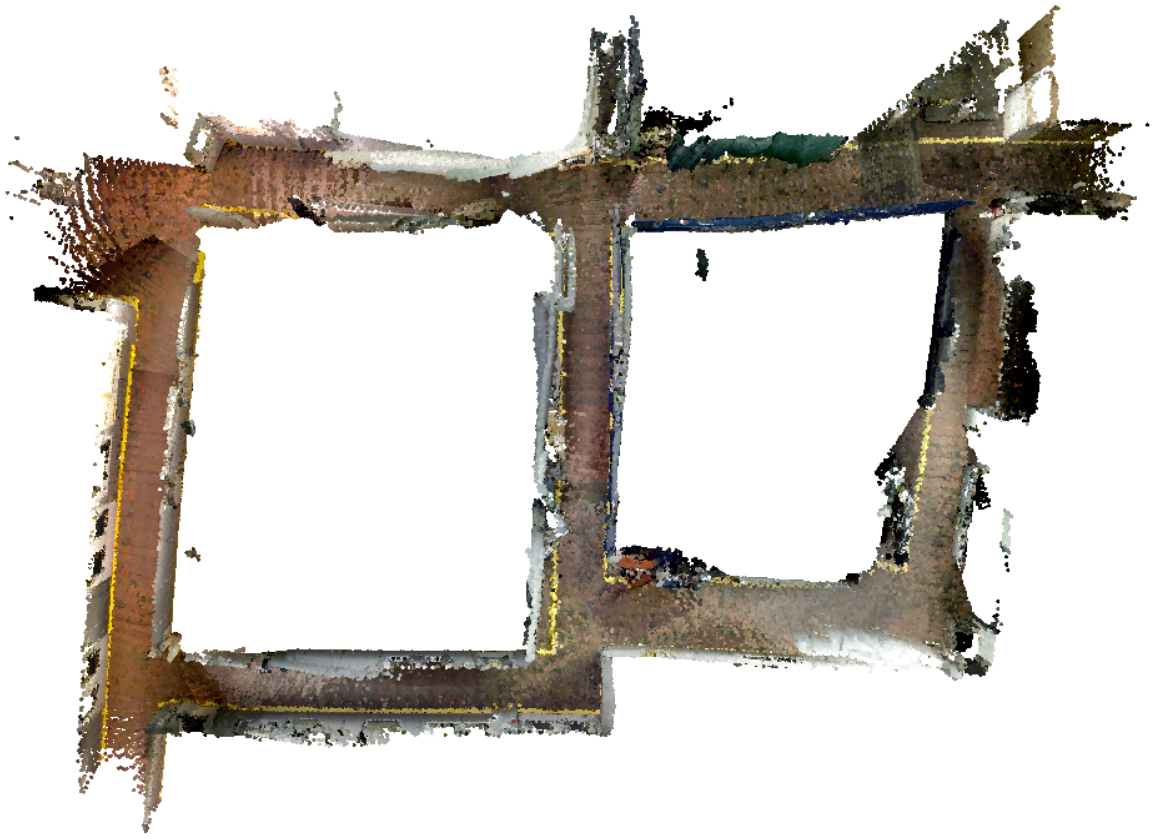


Fig. 7: A point cloud map of a 40m-long corridor. The map was collaboratively built by two robots, and consists of 423 key-frames.

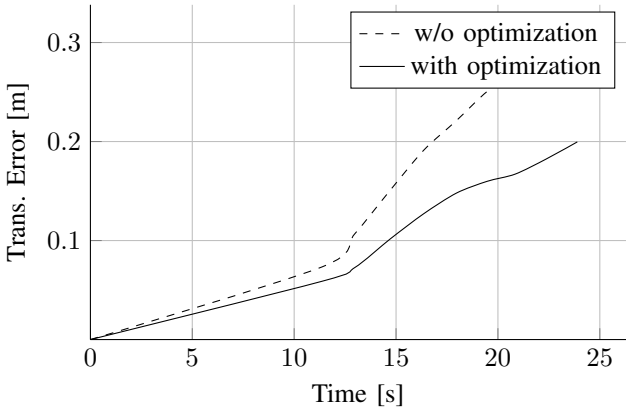


Fig. 12: Translation error of key-frames extracted by visual odometry during a 2 [m] forward motion with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

VII. CONCLUSION

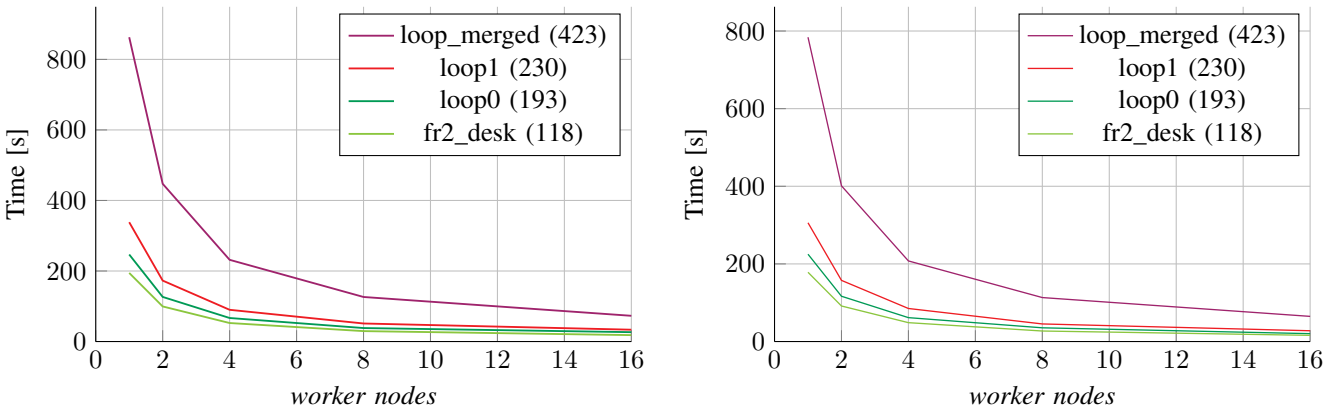
We presented first steps towards a scalable cloud robotics service for mapping and localization using Rapyuta [2], an open-source cloud robotics framework we developed in our previous work.

First, we provided an open source implementation of a state-of-the art, dense visual odometry algorithm on a smartphone-class ARM multi-core CPU[1]. Second, we developed a data protocol that sends only compressed key-frames to the cloud, reducing bandwidth requirements by a factor of two over previous approaches [11]. In addition, the protocol allows the cloud processes to push key-frame pose updates back to the robots without the need for constant polling. Third, we presented techniques for parallelizing the computationally expensive operations of map optimization and map merging in a commercial data center, and provided a corresponding open source software implementation[1].

As illustrated by our demonstrator, this cloud-based architecture holds the potential to greatly increase the number of mobile robot platforms capable of creating large, high-quality maps and performing accurate localization. The robots used were entirely built using low-cost, off-the-shelf components, i.e., an Odroid-U2 board (USD 90), a PrimeSense CARMIN RGB-D sensor (USD 200), a simple iRobot Create robot base (USD 220), and a USB wireless device (USD 40)). Further, the commercial cloud-infrastructure provides computational services at very low cost (USD 0.130 per hour for every *m1.medium* instance ($\sim$2$\times$1.7 GHz, 3.75 GB) [19].

Finally, we showed both qualitative and first quantitative results achieved with the architecture. As shown in Figs. 1 and 7 as well as in the accompanying video, our implementation yields maps comparable to those obtained with more expensive robot hardware. First quantitative experiments confirmed

[1]http://github.com/IDSCETHZurich/rapyuta-mapping



(a) MySQL (b) MongoDB

Fig. 13: Map optimization times against the number of *worker nodes*. The numbers in parenthesis in the legend denote the number of key-frames. loop0 and loop1 are the two loops of the corridor shown in Fig. 7. loop_merged is a combination of both. fr2_desk is a public data set obtained from [29].

that bandwidth requirements are well within those typically available in modern wireless networks (< 0.5 [MB/s]). They also confirmed that map optimization provided via the cloud significantly reduces uncertainty of the robot's visual odometry. Moreover, they confirmed the computational advantage of parallelization for map optimization in the cloud.

Possible future improvements include the incorporation of the depth error into visual odometry [21], substituting the current naive bag-of-words-based place recognition to a more probabilistic approach such as FAB-MAP [30] for map merging, and the creation of larger maps using more robots.

ACKNOWLEDGMENT

This research was funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 248942 RoboEarth. This work also received support from AWS (Amazon Web Services) in Education Grant award. The authors would like to thank Dejan Pangercic for his continuous support and motivation, and their colleagues Dominique Hunziker and Dhananjay Sathe for their support with Rapyuta.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Special Publication 800-145, 2014, available http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] D. Hunziker, G. Mohanarajah, M. Waibel, and R. D'Andrea, "Rapyuta: The RoboEarth cloud engine," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA), (Karlsruhe, Germany)*, May 2013, pp. 438–444.

[3] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *Robotics Automation Mag., IEEE*, vol. 18, no. 2, pp. 69–82, June 2011.

[4] R. Arumugam, V. R. Enti, B. Liu, X. Wu, K. Baskaran, F. K. Foong, A. S. Kumar, D. M. Kang, and W. K. Goh, "Davinci: A cloud computing framework for service robots." in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2010, pp. 3084–3089.

[5] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud Networked Robotics," *IEEE Network*, vol. 26, no. 3, pp. 28–34, May-June 2012.

[6] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2013, pp. 4263–4270.

[7] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2012, pp. 576–583.

[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE Sym. Mass Storage Systems and Technologies (MSST)*. IEEE, May 2010, pp. 1–10.

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

[11] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, 2013, accepted for publication.

[12] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. IEEE Int. Symp. Mixed and Augmented Reality*, October 2009, pp. 83–86.

[13] R. Goransson, A. Aydemir, and P. Jensfelt, "Kinect@Home: Crowdsourced RGB-D data," in *IROS Workshop on Cloud Robotics*, 2013. [Online]. Available: http://www.kinectathome.com/

[14] "Matterport," 2013. [Online]. Available: http://matterport.com/

[15] A. Howard, S. Gaurav S., and M. Maja J., "Multi-robot mapping using manifold representations," in *Proc. IEEE Int. Conf. Robotics Automation (ICRA)*, vol. 4, May 2004, pp. 4198–4203.

[16] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE Tran. Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2013.

[17] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Camera-imu-based localization: Observability analysis and consistency improvement," *Int. J. Rob. Res.*, vol. 33, no. 1, pp. 182–201, Jan. 2014.

[18] Google Inc., "Project Tango," 2014. [Online]. Available: http://www.google.com/atap/projecttango/

[19] Amazon.com Inc., "Amazon Elastic Compute Cloud," 2013. [Online]. Available: http://aws.amazon.com/ec2/am

[20] F. Steinbrucker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense RGB-D images," in *ICCV Computer Vision Workshop*, November 2011, pp. 719–722.

[21] C. Kerl, J. Sturm, and D. Cremers, "Robust Odometry Estimation for RGB-D Cameras," in *Proc Int. Conf. Robotics and Automation (ICRA)*, May 2013, pp. 3748–3754.

[22] T. Tykkala, C. Audras, and A. I. Comport, "Direct iterative closest point for real-time visual odometry," in *ICCV Computer Vision Workshop*, November 2011, pp. 2050–2056.

[23] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," 2004. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf

[24] J. Reinders, *Intel Threading Building Blocks*, 1st ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[25] S. Lovegrove and A. J. Davison, "Real-time spherical mosaicing using

whole image alignment," in *Proc. 11th Euro. Conf. Computer Vision*, September 2010, pp. 73–86.
[26] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2011, pp. 3607–3613.
[27] S. Agarwal and K. Mierle, "Ceres solver," 2014. [Online]. Available: http://code.google.com/p/ceres-solver/
[28] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proc. IEEE Int. Symp. Mixed and Augmented Reality (ISMAR)*, October 2011, pp. 127–136.
[29] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. Int. Conf. on Intelligent Robot Systems (IROS)*, Oct. 2012.
[30] M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

Gajamohan Mohanarajah is a Ph.D. candidate at ETH Zurich supervised by Prof. Raffaello D'Andrea and co-supervised by Prof. Andreas Krause. He obtained his undergraduate and masters degree from Tokyo Institute of Technology, Japan specializing in control and systems engineering. His current interests include cloud robotics, controls, and large-scale machine learning.

Vladyslav Usenko received a B.Sc degree in Computer Science from Kiev National Taras Shevchenko University in 2011 and M.Sc in Informatics from Technical University of Munich in 2013. He is currently a Ph.D candidate at Technical University of Munich specializing in robotics, computer vision and cloud technologies.

Mayank Singh received a B.E.(Hons.) degree in Computer Science from the Birla Institute of Technology and Science, Pilani, India in 2013. He did his Bachelor's thesis at ETH Zurich on Rapyuta, the RoboEarth Cloud Engine, under the supervision of Prof. Raffaello D'Andrea. He is currently a software developer with Cisco Systems, Bangalore, India.

Raffaello D'Andrea received a B.Sc. degree in Engineering Science from the University of Toronto in 1991, and M.S. and Ph.D. degrees in Electrical Engineering from the California Institute of Technology in 1992 and 1997, respectively. He held positions as assistant professor, and later, associate professor at Cornell University from 1997 to 2007. He is currently a full professor of dynamic systems and control at ETH Zurich, and technical co-founder and chief technical adviser at Kiva Systems. A creator of dynamic sculpture, his work has appeared at various international venues, including the National Gallery of Canada, the Venice Biennale, Ars Electronica, and ideaCity.

Markus Waibel received an MSc in Physics from the Technical University of Vienna in 2003 and a PhD in Robotics from the EPF Lausanne in 2007. He is currently a Senior Researcher at ETH Zurich and Program Manager of the cloud robotics project RoboEarth. He is also the co-founder of the ROBOTS Association and its flagship publications Robohub and the ROBOTS Podcast, and the founder of Robotics by Invitation, an online panel discussion of 30 high-profile roboticists.