C^{2}TAM: A Cloud framework for Cooperative Tracking And Mapping

L. Riazuelo, Javier Civera and J. M. M. Montiel

Robotics, Perception and Real-Time Group, Aragón Institute of Engineering Research (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain.

Abstract

The Simultaneous Localization And Mapping by an autonomous mobile robot –known by its acronym SLAM– is a computationally demanding process for medium and large-scale scenarios, in spite of the progress both in the algorithmic and hardware sides. As a consequence, a robot with SLAM capabilities has to be equipped with the latest computers whose weight and power consumption might limit its autonomy.

This paper describes a visual SLAM system based on a distributed framework where the expensive map optimization and storage is allocated as a service in the Cloud, while a light camera tracking client runs on a local computer. The robot onboard computers are freed from most of the computation, the only extra requirement being an internet connection. The data flow from and to the Cloud is low enough to be supported by a standard wireless connection.

The experimental section is focused on showing real-time performance for single-robot and cooperative SLAM using an RGBD camera. The system provides the interface to a map database where: 1) a map can be built and stored, 2) stored maps can be reused by other robots, 3) a robot can fuse its map online with a map already in the database, and 4) several robots can estimate individual maps and fuse them together if an overlap is detected.

Keywords: SLAM, Visual SLAM, Cloud SLAM, Cloud Robotics, Cloud Computing

1. Introduction

The acronym *SLAM*, standing for Simultaneous Localization and Mapping, refers to the problem of simultaneously estimating a model of the surroundings of a mobile robot –the "map"– and the robot's location into it from a stream of sensor data [1]. SLAM is a problem of key importance in robotics; as an accurate model of the environment is a prerequisite of most of the mobile robots' tasks (e.g., navigation, exploration or manipulation). In a practical robotic setting, the computation and memory requirements of the SLAM algorithms are two aspects of prime interest: SLAM algorithms tend to be computationally demanding and the onboard resources of a mobile robot are limited. Also, SLAM has strong real-time constraints as it is integrated in the control loop of the robot.

In recent years the possibility of massive storage and computation in Internet servers –known as *Cloud Computing* and *Cloud Storage*– has become a reality. The availability of such technology and its possible use in robotics have opened the door to a whole new line of research called *Cloud Robotics* [2]. Regarding SLAM, robots could benefit from the use of the Cloud by moving part of the SLAM estimation from their limited computers to external servers; saving computational and power resources. This paper tries to answer the following question *How should a SLAM system be partitioned in order to leverage the storage and computational resources in the Cloud?* Notice that the answer to this question is not trivial. Due to the real-time constraints of SLAM algorithms and the network delays the naïve solution of moving *all* the computation to the Cloud would be unfeasible. In order to guarantee the real-time, part of the computation must be performed on the robot's computers.

The contribution of this paper is the partition of a real-time SLAM algorithm that allows part of the computation to be moved to the Cloud without loss of performance. Our experimental results show that the bandwidth required in all cases does not exceed a standard wireless connection. We demonstrate the capabilities of the framework to provide the interface to a map database in a multi-map multi-camera experiment where the users can: create and save several maps, relocate within them and improve them as new areas are explored, and fuse several maps into one if an overlap is detected. As part of the paper, we plan to release the software after acceptance.

We take as a starting point the monocular SLAM algorithm described in [3], the so-called *Parallel Tracking and Mapping* or *PTAM*. The processing of *PTAM* is based on two parallel threads. On the one hand, a geometric map is computed by non-linear optimization over a set of selected keyframes usually known as Bundle Adjustment. This background process is able to produce an accurate 3D map at a low frame rate. On the other hand, a foreground tracking process is able to estimate the camera location at the frame rate assuming a known map. This method is able to produce maps composed of thousands of points using standard computers for room-sized environments. From this SLAM system, we propose *C^{2}TAM* standing for *Cloud framework for Cooperative Tracking And Mapping* that moves the non-linear map optimization thread to a service operating in the Cloud.

On top of the computational advances of the keyframe based

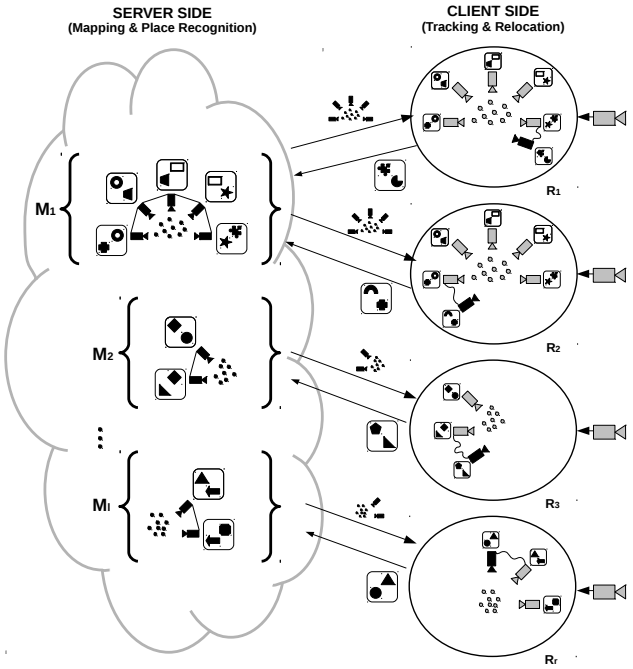


Figure 1: Computer intensive bundle adjustment is performed as a cloud service running on a high performance server. Camera location with respect to the map is computed in low performance mobile devices. Several tracking threads can be run on the same map data. The data flow from tracking to mapping is composed of the new keyframes when gathered images contain new information with respect to the available map; and from mapping to tracking the flow is the computed map.

methods, the resulting communication between the tracking and mapping processes requires low bandwidth. The tracking process sends a new raw keyframe only when the gathered image contains new information with respect to the available map. The mapping sends a new map after every iteration of the Bundle Adjustment, at a frequency substantially lower than the frame rate. Even in exploratory trajectories, the number of new keyframes is small compared with the frame rate; and in the case of already visited areas no new keyframes are sent. See Fig. 1 for a scheme of the framework. The low communication bandwidth allows us to use a standard wireless connection and run tracking and mapping on different computers. Also, in both data flows the algorithm is robust to latencies. The camera tracking thread can run on suboptimal maps until the next global optimization is finished and sent. Also, with an appropriate policy on map management, the camera tracking is robust to delays in keyframe addition.

We believe that a SLAM system partially running on the Cloud has a wide array of benefits and potential applications:

- Allocating the expensive map optimization process out of the robot platform allows a significant reduction in the onboard computational budget, hence reducing the payload and power consumption; both critical factors for field robotics (e.g., unmanned aerial [4] or underwater vehicles). More importantly, it provides the foundations to accommodate SLAM algorithms within the distributed computation framework, which makes it possible to exploit the newly available Cloud computation resources.

- The interoperability between different visual sensors comes as a prerequisite in our system, as very different robots with different cameras could connect to the mapping service.

- The raw keyframe images are stored in the Cloud along with the point-based map. Optimizing a sparse 3D scene of salient points is just one of the Cloud services that can be run over the keyframes. Additionally, other background processes at different time scales can handle map management operations aiming at life-long mapping [5], semantic mapping [6, 7], layout estimation [8] or computing free space for navigation [9].

- The centralized map building also allows a straightforward massive data storage of robotic sequences and geometric estimations that could be used to provide a significant training sample for learning. It can be seen that the size of the computer vision datasets [10] tends to be much larger than those of the robotics [11]. The creation of datasets with data exclusively from robots is essential for exploiting the commonalities in the robotic data [12].

- The proposed framework naturally adapts to the cooperative SLAM problem [13, 14]; where several robots have to build a joint map of the environment. The server can operate on different maps and fuse them independently of the trackers running on the robots. As the number of clients grows the server computation can be parallelized. The bandwidth required for each tracker is low enough to be provided by a standard wireless connection.

- The technologies involved in the proposed system are in an advanced state of maturity: Cloud Computing and storage has already been successfully incorporated in multiple domains and the keyframe-based SLAM is one the most promising mapping methods available [15]. Additionally, the proposed framework can provide the interface to an advanced database of visual maps in the Cloud.

The rest of the paper is organised as follows: Section 2 refers to the related work; section 3 discusses the main SLAM components, section 4 provides the formulation of the SLAM problem and section 5 gives the details of the proposed system, C^2TAM. Finally, section 6 shows the experimental results and section 7 sets out the conclusions and presents lines for future work.

2. Related work

In recent years, the Cloud Computing paradigm has revolutionized almost every field related to computer science [16]. The idea in a few words is that the software pieces are replaced by services provided via the Internet. In the robotics community the potential applications, the benefits and the main lines for research regarding Cloud Computing have already been foreseen [17, 18] and some platforms are already starting to emerge

[19]. Nevertheless, with some recent exceptions referred to in the next paragraph, there is still a lack of specific algorithms and specific realizations of these ideas. This paper aims to contribute with a concrete realization of a SLAM algorithm operating in the Cloud and a thorough experimental testing.

In [20], Bistry et al. analyse how SIFT extraction and matching can be moved from a robot to several servers in the Cloud. [21] queries the Cloud service *Google Goggles* to read text in signs and uses this for loop closing detection. More closely related to our concerns, [22] implements a laser-based FastSLAM algorithm running on the Cloud by distributing the particles among several computing nodes.

Very recently, [23] presented a real-time dense 3D reconstruction that shares some similarities with our approach. Specifically, this work also builds on *PTAM* and uses a distributed architecture very similar to the one presented in this paper. Our contribution is to address the multi-user multi-map case ([23] is a single-user single-map system). The contributions that allow our system to support multiple users and multiple maps are 1) a two-step map relocation robust to communications delays, and 2) a map fusion algorithm that operates in the Cloud server.

In [24, 25], Castle et al. introduce in *PTAM* the relocation capability in a set of multiple maps. Our relocation algorithm is built on top of this work, our main contribution being the two-step relocation that is able to cope with the standard network delays of Cloud Computing.

3. A discussion on the SLAM components

The aim of this paper is to provide a splitting of the SLAM problem such that its strong real-time constraints are not affected by the network delays when Cloud Computing is used. For the sake of clarity in later sections, we will start with a high-level definition of the main components of a modern visual SLAM system.

1. **Mapping.** The mapping component estimates a model of the scene –a map– from sensor data. We will denote the map model as $\mathcal{M}$; and will assume that our mapping system can have several independent maps The initial approaches to SLAM used to update the camera pose and map estimation jointly and sequentially for *every* sensor data arriving [1]. Nevertheless, recent research [3, 15] has proposed a clever partition of the problem into a one frame-rate thread for camera pose estimation and a second one at a lower rate for map estimation, and has shown that it has computational advantages without a loss of performance or accuracy. This is hence the approach we will take in this paper.
2. **Tracking.** The tracking component estimates the camera pose $\mathcal{T}^t$ for every time step t given an estimation of the map $\mathcal{M}$. A multiple-user-multiple-map SLAM system has one tracking component per user. As this pose estimation is based on the tracking of visual features in an image sequence, it should be done at a high frequency and with *strong real-time constraints*. If real-time is lost, the image tracking is likely to fail and hence the pose tracking will also fail.
3. **Relocation.** Once the tracking component has failed, the relocation component tries to relocate the camera and re-start the tracking. The tracking failure can be caused by several reasons: occlusions, high-acceleration motion, blur or lack of visual features. This component also has strong real-time constraints. If the relocation takes too much time the camera might have moved from the relocation position and the tracking component might not be able to start.
4. **Place Recognition.** We understand by place recognition the capability of a SLAM system to relocate in a large number of maps. This problem is also known in the robotics community as the *kidnapped robot problem*; where a robot perceives an unknown environment and has to recognise the place it is in from a number of possibilities. Notice that the difference with relocation is the a priori knowledge on the location. Relocation comes just after a tracking failure, so we can assume a small camera motion and check nearby places to re-start the tracking. Place recognition does not assume any particular location, so every possible location is equally likely and every map in the database should be checked.
5. **Map Fusion.** Map fusion merges two independent maps into one when an overlapping area is detected by place recognition. First, we search for correspondences between local features in the two independent maps. After that, using the geometric constraints of the corresponding points, the rigid transformation between the two maps is computed. One of the maps is then put in the reference frame of the second one and duplicated points are deleted.

4. The SLAM formulation as Tracking and Mapping

4.1. Mapping

The mapping component contains l local maps $\{\mathcal{M}_1, \ldots, \mathcal{M}_k, \ldots, \mathcal{M}_l\}$. Each local map $\mathcal{M}$ is composed of a set of n 3D points $\{\mathcal{P}_1, \ldots, \mathcal{P}_i, \ldots, \mathcal{P}_n\}$ and m keyframes $\left\{C_1, \ldots, C_j, \ldots, C_m\right\}$

$$\mathcal{M} = \left\{\mathcal{P}_1, \ldots, \mathcal{P}_i, \ldots, \mathcal{P}_n, C_1, \ldots, C_j, \ldots, C_m\right\} . \quad (1)$$

Each map entity is modeled with a set of geometric parameters and, for most of them, an appearance descriptor. The model for a 3D point $\mathcal{P} = \left\{\mathbf{P}, d_p\right\}$ contains its Euclidean 3D position $\mathbf{P} = \left(X^W Y^W Z^W\right)^\top$ and a normal $\mathbf{n} = \left(n_x^W n_y^W n_z^W\right)^\top$ in a world reference frame W; and its descriptor $d_p = \{w_1, \ldots, w_r\}$ is composed of r different patches extracted at different scales from a source image. For efficiency reasons, at the implementation level we save the pyramid at r different scales of the source image. Hence, each point $\mathcal{P}$ contains a pointer $(u\ v\ r)^\top$ to a pixel and a scale of the pyramid where the descriptor can be extracted.

A keyframe $C = \{\mathbf{C}, d_c\}$ is modeled quite similarly. First the 3D camera pose $\mathbf{C} = \left(X^W Y^W Z^W \alpha^W \beta^W \gamma^W\right)^\top$ using its Euclidean coordinates $\left(X^W Y^W Z^W\right)$ and roll-pitch-yaw angles $\left(\alpha^W \beta^W \gamma^W\right)$ all of them in a world reference frame W. As the keyframe descriptor d_c we use –as [26]– the frame I_c; subsampled to size 40×30, filtered with a Gaussian mask $g(\sigma)$ and normalized by subtracting the mean.

$$d_c \quad = \quad I_c^{40\times30} * g(\sigma) - \overline{I_c^{40\times30} * g(\sigma)} \, . \tag{2}$$

For each point $\mathcal{P}_i$ we have several image measurements in different keyframes C_j that we will denote as $\mathbf{z}_i^j$. Each image measurement $\mathbf{z}_i^j$ puts a geometric constraint between the geometric parameters of the point $\mathbf{P}_i$ and the keyframe $\mathbf{C}_j$ given by the projection model $\mathbf{f}$

$$\mathbf{z}_i^j \quad = \quad \mathbf{f}(\mathbf{P}_i, \mathbf{C}_j, \mathbf{K}_j) \, ; \tag{3}$$

where $\mathbf{K}_j$ is the internal calibration of the j^{th} keyframe.

The mapping component computes the Maximum Likelihood Estimation (MLE) of the geometric map parameters $\left(\hat{\mathbf{P}}_i, \hat{\mathbf{C}}_j\right)^\top$ by minimising a robust cost function of the error $\Delta\mathbf{z}_i^j$; following what it is usually known as Bundle Adjustment [27].

$$\left(\hat{\mathbf{P}}_i, \hat{\mathbf{C}}_j\right)^\top \quad = \quad \arg\min_{\mathbf{P}_i, \mathbf{C}_j} \sum_{i=1}^{n} \sum_{j=1}^{m} \rho(\Delta\mathbf{z}_i^j / \sigma) \, . \tag{4}$$

The error $\Delta\mathbf{z}_i^j = \mathbf{z}_i^j - \mathbf{f}(\mathbf{P}_i, \mathbf{C}_j, \mathbf{K}_j)$ is the difference between the actual image measurements $\mathbf{z}_i^j$ and the projected ones $\mathbf{f}(\mathbf{P}_i, \mathbf{C}_j, \mathbf{K}_j)$. σ is a median-based estimation of the standard deviation of the measurement's noise. As in [3], in order to avoid the influence of outlier correspondences, we do not minimize the error directly but use a robust function of the error. We use Tukey's biweight function that is defined as

$$\rho(\xi) = \begin{cases} 1 - (1 - \xi^2)^3, & |\xi| <= 1 \\ 0, & \text{else} \end{cases} \; ; \tag{5}$$

The initialization of a map is one of the key aspects in any visual SLAM system [28, 29]. In the first frames of a sequence the *SLAM* estimation is degenerate or quasi-degenerate and hence it might fail quite often. As the *PTAM* system [3] is oriented to Augmented Reality applications, it initializes the map by asking the user to perform a careful translation of the camera in a scene with a dominant plane. We believe that this initialization procedure is not suited to a general robotic application; as we cannot guarantee that the initial motion is a translation –it will be constrained by the scene– or that the scene has a dominant plane.

For the initialization of the proposed *C²TAM* map, we have used an initial multiple model filtering scheme similar to [28]. Filtering schemes are less sensitive to initialization problems in image sequences than approaches based on pairwise correspondences. This initialization process is as follows: an interacting multiple model scheme (see [28] for details) is run on the robot client for the first frames of the sequence. Once enough parallax has been detected and the estimation is not degenerate, the first frame of the sequence is set as the first keyframe C_1 and the current frame as the second one C_2. Both keyframes are sent to the mapping component and the optimization described above is started.

All the experiments in section 6 were run using this initialization. While the initialization process will have an extra computation cost on the client side; it will not be high as the map is just started and its size is small. See the details in section 6.1.

4.2. Tracking

The tracking component models each frame $\mathcal{I}^t$ from the image sequence as a set of geometric and appearance parameters $\mathcal{T}^t = \{\mathbf{T}, d_t\}$. The geometric parameters are those of the camera pose that acquired the frame $\mathbf{T} = \left(X^W Y^W Z^W \alpha^W \beta^W \gamma^W\right)^\top$. The frame descriptor is composed of a global descriptor d_t^G and a set of b local descriptors d_t^L: $d_t = \left\{d_t^G, d_t^{L_1}, \ldots, d_t^{L_b}\right\}$. The global descriptor d_t^G is a subsampled, filtered and normalized version of the frame. The local descriptors d_t^L are the image patches surrounding a set of salient FAST features [30] extracted at 4 scales. Again, for efficiency reasons, only the image pyramid and the FAST feature positions are stored instead of the image patches.

The tracking component estimates the camera pose parameters $\mathbf{T}^t = \left(X^W Y^W Z^W \alpha^W \beta^W \gamma^W\right)^\top$ at every time step k from the information of previous camera poses, the current frame $\mathcal{I}^t$ and the current map $\mathcal{M}_k$. This estimation is done in 3 steps. First, the camera pose $\mathbf{T}^{t|t-1}$ at time t is predicted from the information up to the previous frame at $t-1$ applying a constant velocity model.

In the second step, the points in map $\mathcal{M}_k$ extracted at the coarsest scale –that we will name as $\mathcal{P}_k^*$– are projected into the current image $\mathcal{I}^t$. For each point $\mathcal{P}_{k,i}^*$ we search for its correspondence among the closest salient points in $\mathcal{I}^t$. The camera pose is roughly estimated from this first set of correspondences by a robust minimization.

$$\hat{\mathbf{T}}^{t*} \quad = \quad \arg\min_{\mathbf{T}^t} \sum_i \rho(\Delta\mathbf{z}_i^* / \sigma^*) \, ; \tag{6}$$

where $\rho(\xi)$ is again Tukey's biweight function (equation 5), σ^* is a median-based estimation of the standard deviation of $\Delta\mathbf{z}_i^*$, and $\Delta\mathbf{z}_i^*$ is the reprojection error of each point $\mathcal{P}_{k,i}^*$ in the current frame $\mathcal{T}^t$ at time step t

$$\Delta\mathbf{z}_i^* \quad = \quad \mathbf{z}_i^* - \mathbf{f}(\mathbf{P}_{k,i}^*, \mathbf{T}^t, \mathbf{K}_t) \, . \tag{7}$$

Using this first estimation $\hat{\mathbf{T}}^{t*}$ as a seed, a fine grain estimation for the pose $\hat{\mathbf{T}}^t$ is finally obtained by projecting *every* map point –at every scale and not just the coarsest one– into the current frame $\mathcal{T}^t$ and minimising the reprojection error.

$$\hat{\mathbf{T}}^t = \arg\min_{\mathbf{T}^t} \sum_i \rho(\Delta\mathbf{z}_i/\sigma) \,. \quad (8)$$

Again $\rho(\xi)$ is Tukey's biweight function, σ the median-based estimation of the standard deviation of $\Delta\mathbf{z}_i$, and $\Delta\mathbf{z}_i$ the reprojection error of each point $\mathcal{P}_{k,i}$ in the current frame $\mathcal{T}^t$.

A multiple-user PTAM-like system will have a pose tracking per user; hence the tracking component will be $\{\mathcal{T}_1, \ldots, \mathcal{T}_e, \ldots, \mathcal{T}_r\}$.

4.3. Relocation

Relocation, or the ability to quickly compute the pose of the camera when the tracking thread is lost, is done as a two step process. First, for each new image $\mathcal{I}^t$, its global descriptor d_t is extracted as shown in equation 2. We extract the closest keyframe C_j in the current map $\mathcal{M}_k$ by computing the Euclidean distance between d_t and the global descriptors for each keyframe $d_{c1}, \ldots, d_{cj}, \ldots, d_{cm}$. Assuming that the 3D space is densely populated with keyframes, the nearest-neighbour of d_t will be a keyframe very close in the 3D space.

Using a simple global descriptor, like the reduced version of the image that we use, might seem at first sight to offer a poor representation of the image content. Nevertheless, several works have proved the good performance of such descriptors. [31] shows the most relevant work on the use of downsampled and filtered images as descriptors for scene recognition as a very efficient alternative to more elaborate models with no noticeable degradation in performance. Such a descriptor is called here *Tiny Image*. The key here is the dense population of the image space by growing the training set size to 80 million images. Recently, [32] has reached the same conclusion for the problem of place recognition in mobile robots. The good performance of this descriptor for place recognition in SLAM is also reported in [26]. In these two latest references, the key aspect is again the dense sampling of the image space. In [32] the experiments are done with an autonomous car that shows limited viewpoint differences. In [26] the amount of keyframes in the optimization is kept high to guarantee that a close match will always exist.

After that, the camera location of $\mathcal{I}^t$ is set as that of the keyframe C_j and the rotation is compensated by minimising the error between the global descriptors

$$\left(X^W_{\mathcal{T}^t} Y^W_{\mathcal{T}^t} Z^W_{\mathcal{T}^t}\right)^\top = \left(X^W_{C_j} Y^W_{C_j} Z^W_{C_j}\right)^\top \quad (9)$$

$$\left(\alpha^W_{\mathcal{T}^t} \beta^W_{\mathcal{T}^t} \gamma^W_{\mathcal{T}^t}\right)^\top = \arg\min_{\left(\alpha^W_{\mathcal{T}^t} \beta^W_{\mathcal{T}^t} \gamma^W_{\mathcal{T}^t}\right)} \left(d_t - \mathbf{w}(d_{cj}, \alpha^W_{\mathcal{T}^t} \beta^W_{\mathcal{T}^t} \gamma^W_{\mathcal{T}^t})\right) \quad (10)$$

where $\mathbf{w}(d, \alpha, \beta, \gamma)$ is the warping of the image descriptor d by a rotation given by the angles (α, β, γ). After this initial pose has been assigned, the tracking thread of section 4.2 is re-started. If the tracking is successful the camera is relocated, if not the relocation algorithm of this section is repeated again with the next image $\mathcal{I}_{t+1}$.

4.4. Place recognition and ego-location

By place recognition, we understand the ability to recognise a part of a map $\mathcal{M}_k$ from the visual information in a frame $\mathcal{I}^t$. The subtle difference with the relocation described in section 4.3 is the scale of the problem. The relocation component starts when the camera tracking is lost; hence we can assume that the camera has not moved much and we are in the surroundings of the latest camera pose. Only the closest keyframes in the current map $\mathcal{M}_k$ will be analysed for similarities. The place recognition component, from the information in the frame $\mathcal{I}^t$, tries to recognise a map from all the maps in the SLAM database $\{\mathcal{M}_1, \ldots, \mathcal{M}_k, \ldots, \mathcal{M}_l\}$.

For place recognition and ego-location we will use the same algorithm as in the previous section; but the fact that the computation is larger will introduce differences in our *C²TAM* algorithm, as is detailed in section 5.5. For very large map databases, an interesting line for future work would be to use more efficient search algorithms, such as the Approximate Nearest Neighbour [33] or the recent [34].

4.5. Map fusion

Suppose that, from the l local maps in the Cloud server $\{\mathcal{M}_1, \ldots, \mathcal{M}_k, \ldots, \mathcal{M}_q, \ldots, \mathcal{M}_l\}$, the place recognition component from section 4.4 has detected that maps $\mathcal{M}_k$ and $\mathcal{M}_q$ overlap in some specific region. The map fusion component merges the two maps in a common reference frame.

Our map fusion algorithm works as follows. When the map optimization over $\mathcal{M}_k$ receives a new keyframe C^k_j from the tracking node; the latest is compared with every keyframe in the rest of the maps in the server. Similarly to the relocation component of section 4.3, we will use the global descriptor d_t as defined in equation 2 to quickly extract a set of potential keyframe candidates that are imaging the same area.

In a second step, once we have two potentially overlapping keyframes C^k_j and C^q_h from the maps $\mathcal{M}_k$ and $\mathcal{M}_q$ we search for point correspondences between the two maps. We project the 3D points from the two maps $\mathcal{P}^k$ and $\mathcal{P}^q$ in the common keyframe C^k_j, resulting in two sets of image points $\mathbf{z}^{j,k} = (\mathbf{z}^{j,k}_1, \ldots, \mathbf{z}^{j,k}_{i_k}, \ldots, \mathbf{z}^{j,k}_{n_k})^\top$ and $\mathbf{z}^{j,q} = \left(\mathbf{z}^{j,q}_1, \ldots, \mathbf{z}^{j,q}_{i_q}, \ldots, \mathbf{z}^{j,q}_{n_q}\right)^\top$.

$$\mathbf{z}^{j,k}_{i_k} = \mathbf{f}\left(\mathbf{P}^k_{i_k}, \mathbf{C}_{j,k}, \mathbf{K}_j\right) \quad (11)$$

$$\mathbf{z}^{j,q}_{i_q} = \mathbf{f}\left(\mathbf{P}^q_{i_q}, \mathbf{C}_{j,k}, \mathbf{K}_j\right) \,. \quad (12)$$

As the two keyframes are assumed to be very similar, correspondences between $\mathbf{z}^{j,k}_{i_k}$ and $\mathbf{z}^{j,q}_{i_q}$ are computed based on their distance in the image plane $\|\mathbf{z}^{j,k}_{i_k} - \mathbf{z}^{j,q}_{i_q}\|$: If this distance is lower than a threshold (2 pixels in our experiments) the image points are considered to match. Using the correspondences between $\mathbf{P}^q$ and $\mathbf{z}^{j,k}$ –3D points in map $\mathcal{M}_q$ and image projections from map $\mathcal{M}_k$– we can compute the relative transformation between the keyframe C^k_j and the map $\mathcal{M}_q$ using the Perspective-n-Point (PnP) [35]. The relative transformation between the maps $\mathcal{M}_k$ and $\mathcal{M}_q$ are calculated from the composition.

Finally, the duplicated points (correspondences between $\mathbf{z}^{j,k}$ and $\mathbf{z}^{j,q}$) are deleted and the rest of the points and cameras in

map $\mathcal{M}_q$ are transformed according to the relative motion between the two maps and merged into $\mathcal{M}_k$.

It should be noticed that the formulation of the proposed mapping and tracking components in sections 4.1 and 4.2 uses only the *RGB* information; and hence the maps in the database are estimated up to a scale factor. This fact becomes relevant for map fusion, as two maps of different scales cannot be fused directly. There are two possible solutions. The first one is to estimate the transformation and the scale when two maps are fused; as for example in [36]. The second one is, if a *RGB-D* sensor is used, to extract an estimation of the real scale of the map from the depth channel of the camera.

We chose the second option for this paper. For all the points $\{\mathcal{P}_1, \ldots, \mathcal{P}_i, \ldots, \mathcal{P}_n\}$ in a map $\mathcal{M}$ we extracted their real depth measurement $D^{RGB-D} = \left(D_1^{RGB-D}, \ldots, D_i^{RGB-D}, \ldots, D_n^{RGB-D}\right)$ from the *RGB-D* keyframe where they were initialised. We then extracted their depth values at the map scale as the distances $D^{\mathcal{M}} = \left(D_1^{\mathcal{M}}, \ldots, D_i^{\mathcal{M}}, \ldots, D_n^{\mathcal{M}}\right)$ between each point position $\mathbf{P}_i$ and the position of the keyframe C_j where it was initialised. Finally, we estimated the scale ratio as the median value of $D^{RGB-D} - D^{\mathcal{M}}$. With this value, each map can be transformed into a real-scale map and it can be fused with any other map in the database using the algorithm described in this section.

Notice that the relocation in section 4.3 and the map fusion in this section allow several working modes:

- Single-user-multiple-maps, where a single user is estimating a map that can be fused with other map instances in the database –previously estimated by other users. As a result, the final map comes from the fusion of two or more maps estimated at different times possibly by different users.

- Multiple-user-multiple-maps, where several users are estimating independent maps at the same time that might be fused if they belong to the same environment. As a result, we obtain a global map from several individual maps that are being estimated at the same time. After the maps are fused, the different users can keep tracking and improving the global map cooperatively.

5. C²TAM: A SLAM in the Cloud

5.1. Mapping as a Cloud Service

The estimation of the map database of our system $\{\mathcal{M}_1, \ldots, \mathcal{M}_k, \ldots, \mathcal{M}_l\}$ is the most demanding computation in our framework and does not have strong real-time constraint. The map optimization in equation 4 can take several frames of the sequence and the tracking can still operate in a non-optimal map from a previous optimization. The mapping component might be a perfect candidate for Cloud Computing as it can tolerate the network delays; but it is also necessary that the data flow with the onboard robot computers is low enough. We will analyse this in the next sections.

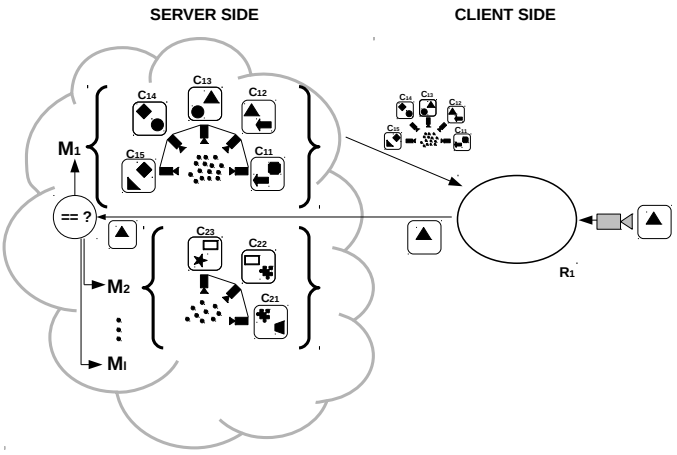


Figure 2: Coarse-grained place recognition in the Cloud server. The robot client $\mathcal{R}_1$ sends a frame from the sequence to the server; which tries to relocalize the camera with respect to every keyframe in every map in the database using the algorithm in section 4.3. If there is a match, a set of close keyframes is downloaded to the client for fine-grained place recognition.

5.2. Tracking as a client in the robot

The camera pose tracking is a process with strong real-time constraints that has to operate at the frame rate and might fail if a few frames are skipped. This component is hence not resilient to network delays and should be allocated in the robot.

The mapping service in the Cloud receives as input new keyframes from the tracking client. This produces a low-bandwidth traffic, as typically the ratio of keyframes to total frames in the sequence is quite low (in our experiments, this ratio is around 10^{-2}). The mapping serves to the client the current map $\mathcal{M}_k$ every time the map is optimized. This produces quite high traffic.

5.3. Relocation as a client in the robot

Relocation refers to the ability of a SLAM system to relocate in a map previously estimated and stored in the Cloud. A tracking node may need to relocate in two cases: (i) the tracking node, operating successfully over a map, is lost because of a sudden motion or large occlusion. In this case, the camera is likely to be in the previous map, and relocation should only check the current map. This relocation is performed on the tracking node. (ii) The tracking thread has been lost for a long time, or just started. In this case, the camera could possibly be in a large number of maps. In this case, relocation should look for correspondences against a possibly very large number of stored maps. As this case will be more demanding, it should run partially on the mapping node.

5.4. Place recognition and ego-location in two steps

Place recognition and ego-location in a large number of maps can be computationally demanding, and hence it should be allocated in principle as a Cloud service. On the other hand, it is also a critical process sensitive to network delays: if the ego-location estimation takes too much time, the robot might have moved and already be in another place. In our experiments, *Tiny Image* comparisons take around 0.03 milliseconds. With maps of several hundreds of keyframes and databases of

possibly hundreds of maps, the cost of this place recognition might easily be a second. This is why we propose a two-stage relocation algorithm, the first part being run on the Cloud and the second in the robot client.

In the first stage, the mapping server in the Cloud coarsely relocates the camera in a possibly large number of maps. This first coarse relocation can take a significant amount of time, and the camera may have moved when the relocation data arrives at the client. The server sends to the client a small number of filtered relocation candidates consisting of the closest keyframe from the first stage and several close keyframes from the same map; assuming that the camera may have moved during the relocation search in the server. Figure 2 illustrates this process.

In the second stage, the client runs a fine-grained relocation among the filtered candidate keyframes sent by the server. Delays are not influential in this second stage, as relocation has a very small number of candidates and it is entirely done by the client without data transmission. A scheme of this fine-grained relocation can be seen in figure 3.

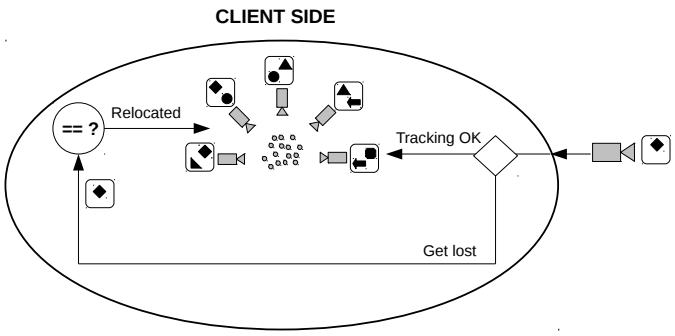


Figure 3: Fine-grained place recognition on the robot client. The robot client tries to relocate with respect to the set of candidates coming from the first stage, using the algorithm from section 4.3

5.5. Map fusion as a Cloud service

The map fusion is entirely done in the Cloud server in our C^2TAM framework; as it is a process that does not have real-time constraints. An illustrative example of the map fusion algorithm in the Cloud and the associated data traffic is shown in figure 4. In figure 4.a a robot client $\mathcal{R}_1$ is tracking the camera pose and uploading a new keyframe C_3^1 to the map $\mathcal{M}_1$. Every new keyframe that is uploaded to the Cloud server is compared with every map in the database. In this case the server detects an overlap of this new keyframe C_3^1 in $\mathcal{M}_1$ with the keyframe C_2^2 in $\mathcal{M}_2$. This is graphically shown in figure 4.b.

The map fusion algorithm from section 4.5 is then applied; map $\mathcal{M}_1$ and $\mathcal{M}_2$ being fused into a single map $\mathcal{M}_{1,2}$ as is shown in figure 4.c. Notice that this fusion might produce high traffic between the Cloud server and the robot client as the keyframes and points that did not have a local copy in the robot client –in this case, the keyframes and points in map $\mathcal{M}_2$– have to be sent. But notice also that this process does not have strong real-time constraints. The robot can track its pose with a suboptimal map while new keyframes are arriving, as is shown in the experiment in section 6.3.

In the case that $\mathcal{M}_2$ is a large map and its download requires a large amount of time, some heuristics can still be applied in order to guarantee the robustness against latencies.

- When a map $\mathcal{M}_k$ is sent from the server to a client, the keyframes $\{C_1, \ldots, C_j, \ldots, C_m\}$ should be downloaded according to their distance from the current camera frame $\mathbf{T}^t$. In this manner the client always has the best possible estimation for the areas that are currently being visited; while far areas might be outdated. We implemented this heuristic in our code and did not observe a big difference in our experiments, but it might be an important technical detail in the case of large maps.

- While a large map $\mathcal{M}_k$ is being downloaded, the client can be exploring new areas and hence uploading new keyframes to the server. The estimated pose of these new keyframes might take a long time to be downloaded by the client, as the large map $\mathcal{M}_k$ has to be downloaded first. The solution for this problem is to perform a local Bundle Adjustment in the server any time a new keyframe arrives, and send it to the client with high priority. In this manner, we are sure that a first initial seed for new keyframes will be sent very quickly to the client and tracking is not lost. Again, we did not encounter this problem in our experiments but it might be a relevant issue for large mapping in the Cloud.

6. Experimental results

In this section we detail 4 experiments that show different modes of use of the proposed C^2TAM framework. All the experiments were recorded using *RGBD* cameras at 640×480. Nevertheless, it should be noted that only the *RGB* channels are effectively used for the visual SLAM and the depth from the *D* channel is only used for visualization. All the experiments were run in real-time and using the standard wireless of our University, demonstrating that the proposed system is resilient to its network delays.

The experimental results are organized as follows. In section 6.1 a single-user-single-map experiment is performed to evaluate the cost and bandwidth required for the operation of the tracking, mapping and relocation components. Section 6.2 shows a single-user-multiple-maps experiment to demonstrate the real-time place recognition capabilities. Once the real-time relocation in a stored map is shown, section 6.3 shows how an online estimation of a map can be fused with a previously stored map. Finally, section 6.4 presents a multiple-user-multiple-map experiment where two independent maps of the same room are first estimated, fused when a significant overlap is detected, and the two users keep enlarging the joint map cooperatively after the fusion.

6.1. Cost and Bandwidth Analysis

In this experiment, a single user is estimating a single map. Our aim is to illustrate the computational advantages of the

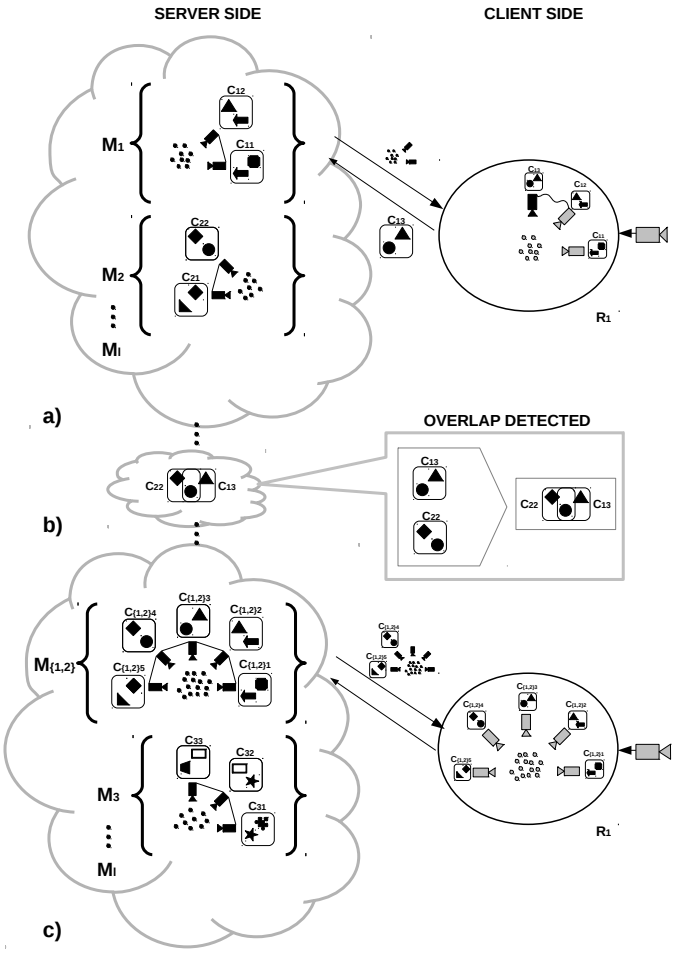


Figure 4: Map fusion in the Cloud. a) The robot client $\mathcal{R}_1$ uploads a new keyframe C_3^1 to the map $\mathcal{M}_1$. b) The keyframe C_3^1 in $\mathcal{M}_1$ presents an overlap with the keyframe C_2^2 in $\mathcal{M}_2$. c) $\mathcal{M}_1$ and $\mathcal{M}_2$ are fused into $\mathcal{M}_{1,2}$ and the fused map is downloaded by the robot client $\mathcal{R}_1$.

proposed architecture with a simple example before going into more elaborate examples. The camera tracking client was run on a laptop (Intel Core i7 M 620 at 2.67GHz, 4GB RAM) and the mapping service was run on a desktop PC (Intel Core i7-2600 at 3.4GHz, 8GB RAM). Both processes, tracking and mapping, have been implemented using ROS (Robot Operating System) [37] and the open source libraries of *PTAM* [3]. The tracking client and the mapping server were connected through the standard wireless connection in our institution.

Figure 5 shows in a double-axis figure the computational cost per frame of the tracking client and the size of the map for a 4961 frame experiment. It can be seen that the tracking cost remains constant at around $10ms$ –even when the map size is growing–, well under the threshold imposed by the frame rate of the sequence that is $33ms$. This suggests that the tracking could be done on a lower-performance computer. Notice that with a proper policy of reducing the number of measured points –as done in [38]– this cost could be even lower and the tracking run on mobile phone devices. Notice also the high computational cost (around $70ms$ per frame) of the first 20 frames of our algorithm. This is caused by the automatic initialization

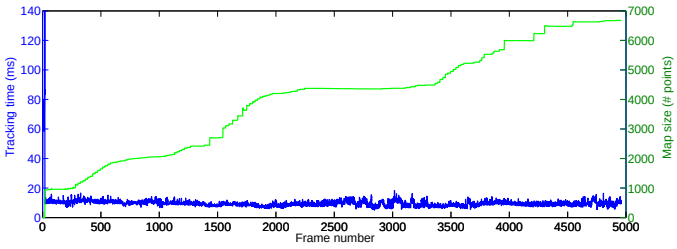


Figure 5: Computational cost of the camera tracking client for a 4961 frame experiment and map size. Notice the almost constant complexity and low cost of the tracking thread, even when the map size grows.

described in section 4.1. This is clearly a line for future work, as we are able to avoid the manual initialization of *PTAM* and the initialization algorithm works at an even lower frame rate. However, the cost is rather high and should be reduced.

Figure 6 shows the bandwidth required for the algorithm in the same experiment. The horizontal axis is set in seconds to compare with the bandwidth of a standard wireless. The blue spikes show the required bandwidth for the data from tracking to mapping, registered at the time the data arrives. Notice that all the blue spikes are of the same height. This is because the only communication from the tracking client to the mapping server comes from uploading keyframes, which have the same size. The red spikes stand for the data flow from the mapping to the tracking. This data flow is the download of the 3D points from the map.

Notice also in figure 6 that the server-to-tracking data flow –the red spikes– slightly increases over time. This comes as a result of the map becoming larger as we explore. While this is not a problem for the experiments described in this paper, it might become a relevant issue for very large maps. In this case, the same solution we proposed in section 5.5 applies: the keyframes and points that are closer to the current camera pose should be given a higher priority for the client not to lose track.

C²TAM succesfully built the map from this sequence using a standard wireless connection. The average bandwidth required was around $1MB/s$; which should be less than the maximum available –a standard number is $3.75MB/s$. We measured the specific bandwidth in our institution and found it to be around $1.45MB/s$; below the maximum but still enough to guarantee the good performance of our algorithm.

With the figures above, we can roughly estimate the delays in the data transmission. New keyframes that are sent from the client to the server are 640×480 *RGB* images, whose size is around $1MB$. At a transmission rate of $1.45MB/s$, the expected delay is approximately 1 second. The data flow from the server to the mapping is dependent on the map size and so are the delays, but for the typical sizes of our experiments we estimated a delay of $100 - 200$ milliseconds. Notice that the two delays are much bigger than the time between camera frames –$1/30$ seconds–; making the naïve approach of sending every frame to the server infeasible and justifying our approach. Notice also that neither the mapping service nor the tracking client was influenced by this network latency in the communications with the proposed *C²TAM*.

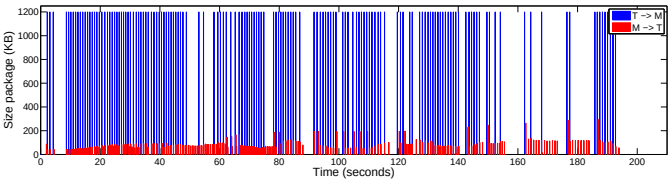


Figure 6: Data flow produced by C^2TAM in a sequence of 4961 frames (around 3 minutes). Red represents data from the mapping service to the tracking client, blue represents data from the tracking client to the mapping server. Each peak is registered at the time the data arrives. The average data flow for this experiment was $1MB/s$, below the usual wireless bandwidth of $3.75MB/s$.

6.2. Relocation in Multiple Maps

In this experiment, three different maps were created using the single-user-single-map mode from the previous section 6.1. The RGBD sequences were recorded in different areas in our research laboratory. The first one, called *desktop sequence*, was recorded in the surroundings of the desktop of one of the authors. The second one, called *wall and bookshelf sequence*, was recorded pointing the camera at a bookshelf. Finally, the third one, called *hospital room sequence*, was recorded in a replica of a hospital room available in our laboratory.

After each map was created, it was saved and stored in the Cloud server. Figures 7(a), 8(a) and 9(a) show some keyframes for each of the sequences; that is the desktop sequence, the wall and bookshelf sequence and the hospital room sequence in that order. Figures 7(b), 8(b) and 9(b) show a 3D view of the estimated maps of the desktop, wall and bookshelf, and hospital room respectively.

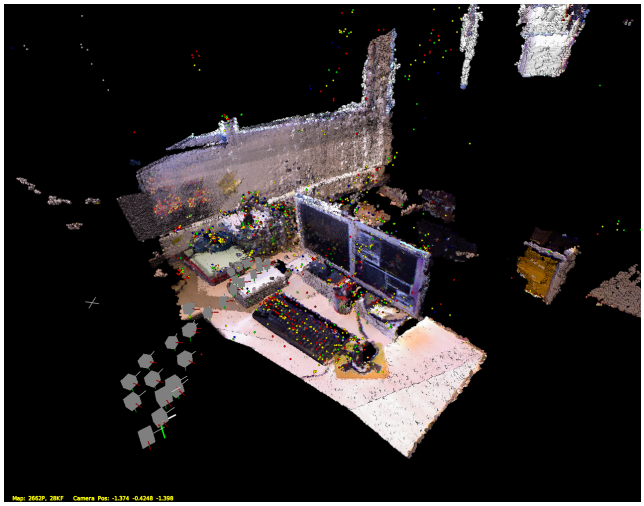
After these three maps were created, another sequence traversing the whole laboratory and hence the three above mentioned areas was recorded. Figure 10 shows some keyframes of this new sequence, named as *laboratory sequence*. Notice that although this new sequence covers the same three scenes, the new keyframes show some areas that were not seen before and hence will be able to extend and improve the previous maps in figures 7(b), 8(b) and 9(b).

In this new sequence, C^2TAM tried to relocate the current camera in one of the three previously stored maps as described in section 4.3.

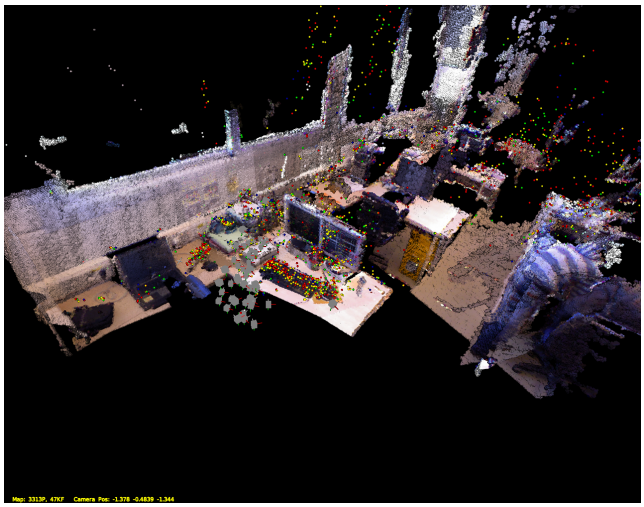
Once the camera was successfully relocated in one of the maps, the tracking thread in the client added new keyframes to this map and then extended and improved this map. Figures 7(c), 8(c) and 9(c) show the improved maps for each of the scenes –desktop, wall and bookshelf, hospital room– respectively. The maps were noticeably extended, as can be seen when compared with the maps before the *laboratory sequence* was processed (figures 7(b), 8(b) and 9(b)). Specifically, the *desktop map* was extended from 2662 points and 28 keyframes to 3313 points and 47 keyframes, the *wall and bookshelf map* from 2711 points and 32 keyframes to 3987 points and 58 keyframes, and the hospital room from 642 points and 29 keyframes to 1624 points and 58 keyframes.

After the relocation and improvement process, a set of measurements has been performed in order to verify the accuracy of the maps generated by C^2TAM. Figure 11 shows the measurements taken, and the Table 1 contains the comparison of these measurements with the ground truth of the scenarios. The ground truth measurements were made by a tape measure with millimeter precision due to the tape resolution.

(a) Sample keyframes from the desktop sequence.



(b) Map estimated from the desktop sequence.



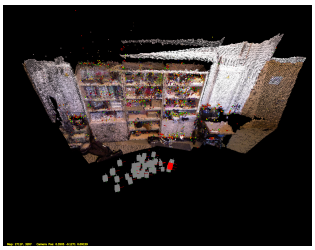
(c) Estimated map for the desktop after the laboratory sequence

Figure 7: Keyframes and map for the desktop scene.

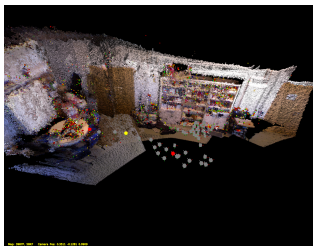
6.3. Overlapping map fusion

This experiment aims to show the map fusion capabilities of the proposed C^2TAM system. The mapping server contains the maps estimated for the previous experiments. We run the C^2TAM on a sequence imaging a desktop. We will denote this scene as *desktop A*. The camera later moves to another desktop that we will call *desktop B*. Figure 12(a) shows the estimated map of *desktop A*, just before moving to *desktop B*.

(a) Sample keyframes from the wall and bookshelf sequence.

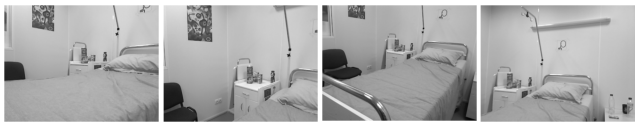


(b) Map estimated from the wall and bookshelf sequence.

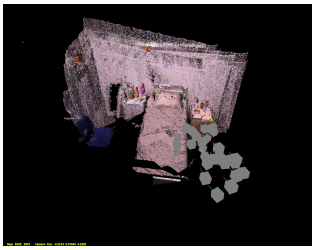


(c) Estimated map for the wall and bookshelf after the laboratory sequence

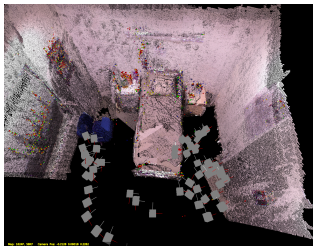
Figure 8: Keyframes and map for the wall and bookshelf scene.



(a) Sample keyframes from the hospital room sequence.



(b) Map estimated from the hospital room sequence.



(c) Estimated map for the hospital room after the laboratory sequence

Figure 9: Keyframes and map for the hospital room scene.

Figure 10: Sample keyframes from the sequence traversing the whole laboratory.

Among the maps in the server there is an estimated map of *desktop B*, which is shown in figure 12(b). As the camera goes from *desktop A* to *desktop B* and adds keyframes to the map, the similarity of the new keyframes and every keyframe in every other map in the server is computed. When the camera reaches *desktop B*, the map fusion process detects the similarity with a keyframe from a previous map (see the keyframes from both maps in figure 13) and merges them into one. Figure 12(c) shows the map when the similarity is detected, and figure 12(d) shows the map after merging with the map in the database. Notice the correct alignment after the map fusion in this latest figure.

Map	Element	C^{2}TAM	ground truth
Desktop	Laptop	0.340	0.337
Desktop	Table	1.107	1.102
Desktop	Window	1.205	1.201
Room	Bed	2.039	2.044
Room	Cabinet	0.418	0.420
Room	Walls	3.105	3.096
Wall	Bookshelf	2.723	2.727
Wall	Door	1.790	1.795
Wall	Table	1.106	1.107
Wall	Whiteboard	1.267	1.264

Table 1: Comparison between C^2TAM measurements and the ground truth. All the measurements are in metres.

Regarding the map fusion process, the communication flow between tracking and mapping is the critical point. C^2TAM deals with this providing an approach that efficiently manages the amount of data exchanged between processes. Once the overlap is detected, the mapper server merges both maps and has to send the new keyframes and points to the tracker.

In the experiment proposed, the current map desktop A (112 keyframes and 7925 points) is fused with a previous map desktop B stored in the server (42 keyframes 4098 points). This process involves sending the new keyframes and points from the server to the client. In this case the data flow generated is 16 MB, according to the standard wireless bandwidth. This communication takes about 5 seconds. This time could be reduced taking into account the principle of locality: if the server starts to send only the information related with the keyframes close to the pose of the current camera, the data flow will be reduced.

Despite the amount of data flow exchanged, the client performance is not affected. The client is working with a suboptimal map while the mapping is sending the new information about the keyframes and points. The client can work properly because the working area is covered by keyframes contained in the suboptimal map. Once new information (keyframes and points) arrives, the client takes into account this information in the tracking process and updates the local copy of the map.

For more details on the experiment in this section, the reader is referred to the video accompanying the paper [1].

6.4. Cooperative SLAM

This section shows a cooperative SLAM experiment using the proposed framework. The aim is mapping an office using several cameras initially unaware of each other. The clients attached to the cameras perform tracking in each map separately.

[1] A high resolution version of the video can be found at `http://webdiis.unizar.es/~riazuelo/videos/c2tam_desktop.mp4`

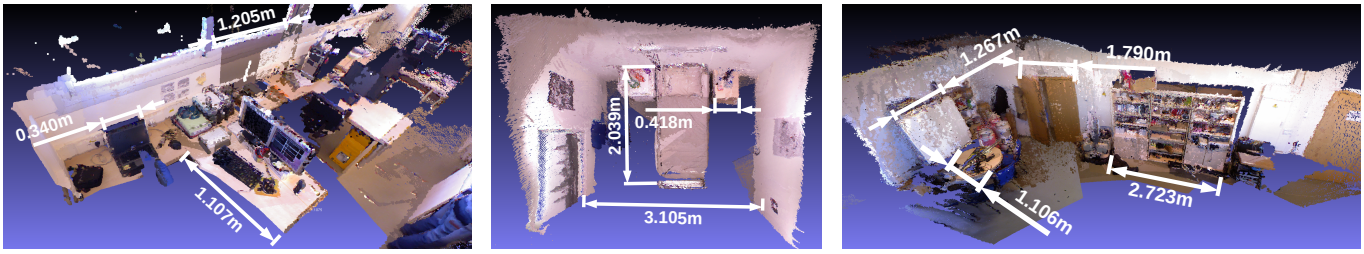
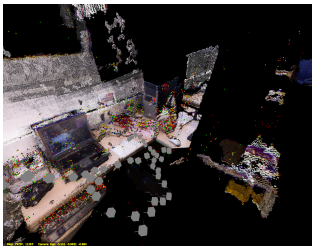
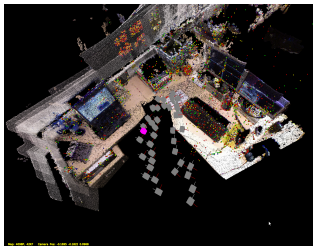


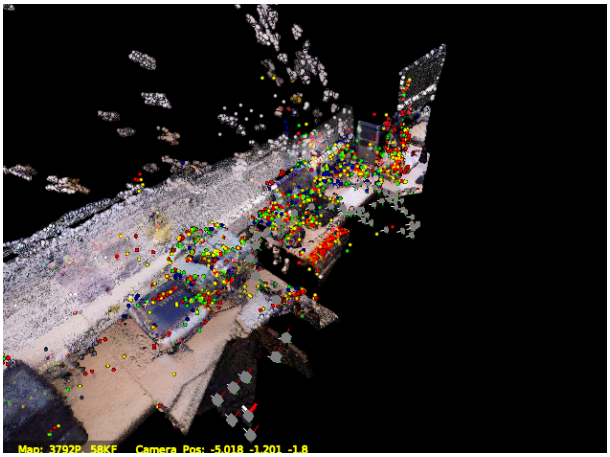
Figure 11: Set of the measurements taken on the scenarios.



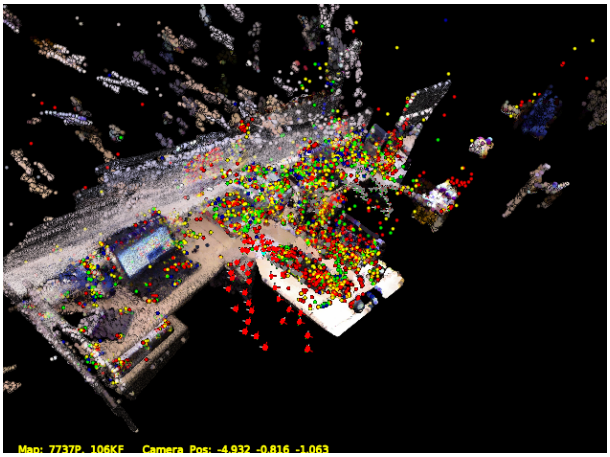
(a) Map of *desktop A*, estimated online.



(b) Map of *desktop B*, stored in the mapping server.



(c) Map of *desktop A* when a common area with the map of *desktop B* is detected.



(d) Maps *desktop A* and *desktop B* after the fusion.

Figure 12: Several snapshots of the maps for the map fusion experiment.

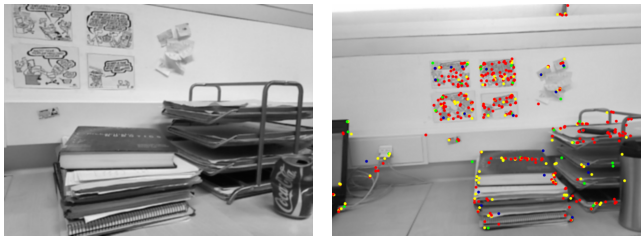


Figure 13: Images of the keyframes of the previous (left) and the actual map (right).

As shown in section 6.3, the server optimizes every map but also looks for overlaps between them. When an overlap is detected, the server fuses both maps. From there, both trackers operate on the new fused map.

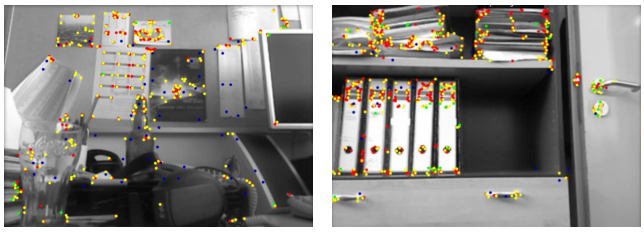


Figure 14: Initial images taken by first tracker (left) and second tracker (right).

Specifically, we used two RGBD cameras and cooperatively mapped an office in real-time. Two laptops (Intel Core i7 M 620 at 2.67GHz, 4GB RAM) were attached to the cameras and acted as clients. A PC (Intel Core i7-2600 at 3.4GHz, 8GB RAM) in another building of our university acted as the server. In several parts of the experiment the trackers were lost and were able to relocalize as explained in section 4.3.

Figure 17 shows the scene to be mapped. Figure 14 shows the initial images taken by the two cameras. Notice that the areas do not overlap and hence two different maps are started: the first one of a desktop and the second one of a bookshelf. Figure 15 shows the 3D estimated maps after some keyframes have been added.

When the second tracker approaches the desktop area, eventually a significant overlap is found. Figure 16 shows the actual image of the second tracker just before the fusion and an image of the first tracker from the beginning of the experiment when this area was mapped. Notice that the image approximately shows the area above the desktop where the first map started.

After the overlap detection and motion estimation, both maps are finally fused. Map 1 contained 25 keyframes and 1578 points, and map 2 38 keyframes and 4833 points. The fused map contains 63 keyframes and 6302 points. Notice that while the number of keyframes is the sum of the individual maps, this does not happen with the number of points as the duplicated ones are deleted. Figure 18(a) shows the individual 3D maps of each tracker before the fusion and figure 18(b) shows the final map after the fusion. After the fusion both trackers map the scene over the same map. The following figure 19 shows the potential of the framework for cooperative SLAM: the second tracker stops mapping (figure 19(b)), and the first tracker continues exploring and expanding the office map (figure 19(a)). Figure 19(c) shows the map before and after the expansion. Finally, when tracker two starts moving again, it is able to relocalize in the area that tracker one has mapped as both are working on the same map instance.

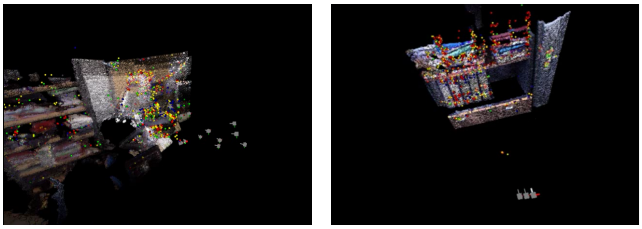


Figure 15: 3D estimated maps by first tracker (left) and second tracker (right).

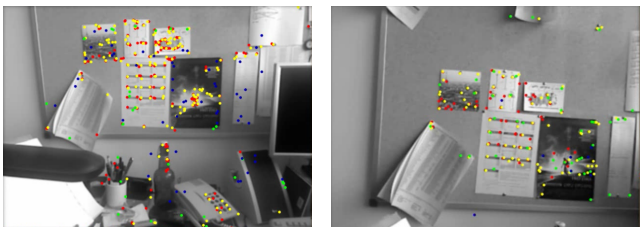
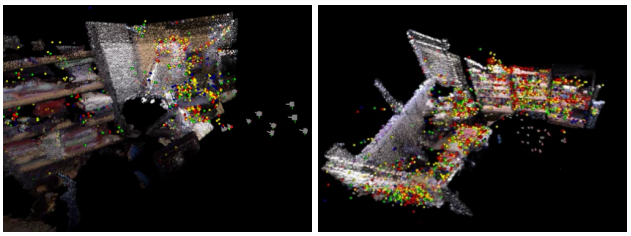


Figure 16: Images of both trackers on the same area. First tracker (left) second tracker (right).

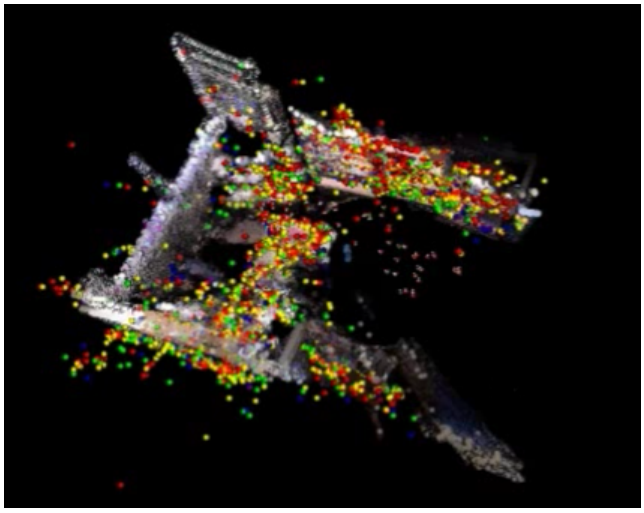
Figure 20 shows the final 3D reconstruction built cooperatively by the two cameras. This final map contains 100 keyframes and 7483 points. We encourage the reader to see the video [2] of the experiment for further understanding.

7. Conclusions and future work

The paper describes a novel framework for distributed keyframe-based SLAM where the map optimization is moved to a server or an array of them outside the robot –the Cloud–. The robot only has to run a light camera tracking and client relocation and has access to an Internet connection. A direct consequence is that the computational load on the client side –on the robot– is reduced. This reduction might be critical in robotic applications with strict constraints on both power and weight, such as unmanned underwater or aerial vehicles.



(a) 3D maps of each tracker before fusion. First tracker (left one) second tracker (right one).



(b) Final map after fusion both maps.

Figure 18: 3D maps before and after fusion process.

Our algorithm exploits the fact that the state-of-the-art visual SLAM algorithms divide the Simultaneous Localization and Mapping problem into two parallel threads, one for camera pose tracking and the other for map optimization. Our experimental results demonstrate that the communication between the two threads is small enough to be supported by a standard wireless, and that the latency introduced by the network does not influence the performance of the algorithm. In our experiments we use an *RGBD* sensor, using the *RGB* images to align the cameras and the point clouds from the *D* channel for visualization and map fusion.

The most critical point of the algorithm where the latency is important is the relocation with previous maps. This paper contributes with a 2-stage relocation algorithm; an expensive coarse relocation is run on the server over all the stored maps and returns the specific map where the camera is, and a second fine relocation downloads this map to the client and runs a cheap relocation on the keyframes of this map. We have also demonstrated the ability to build maps concurrently between several sensors observing the same environment.

We have implemented a map fusion component that enables the possibility of building a cooperative map of an environment. Each robot can explore a new area and estimate a map while

[2] A video of the whole experiment can be found at `http://webdiis.unizar.es/~riazuelo/videos/c2tam_room.mp4`

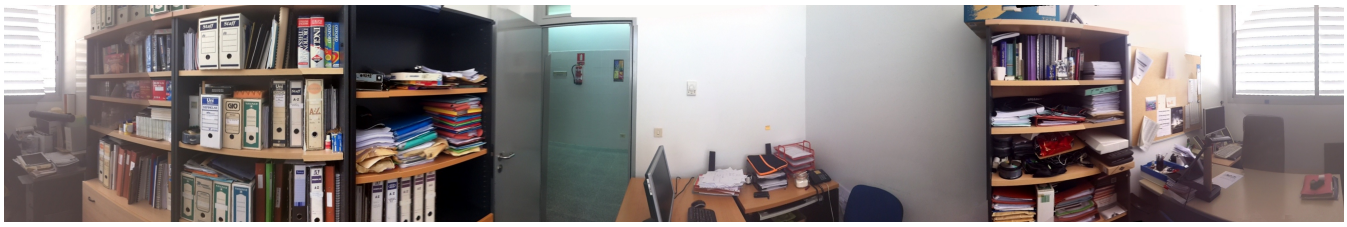
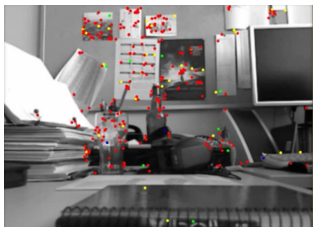
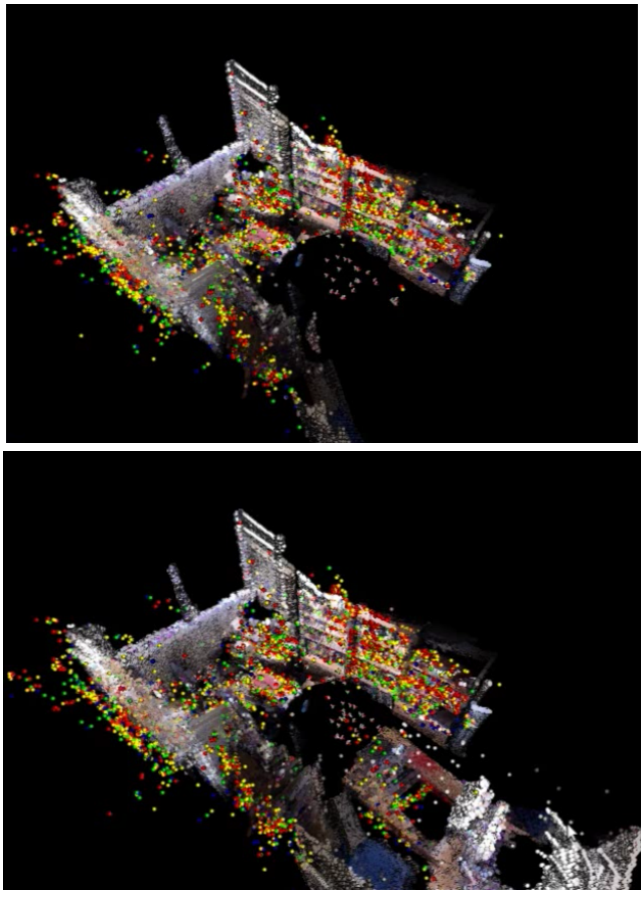


Figure 17: Panoramic snapshot of the experimental environment room.

(a) First tracker image.



(b) Second tracker image.



(c) 3D map before and after the first tracker update

Figure 19: Cooperative update of the map.

the Cloud server looks for similarities with the maps that the rest of the robots are estimating. Once the map of one robot is detected to have a common area with another map, the Cloud server will fuse both into a single one independently from the tracking processes.

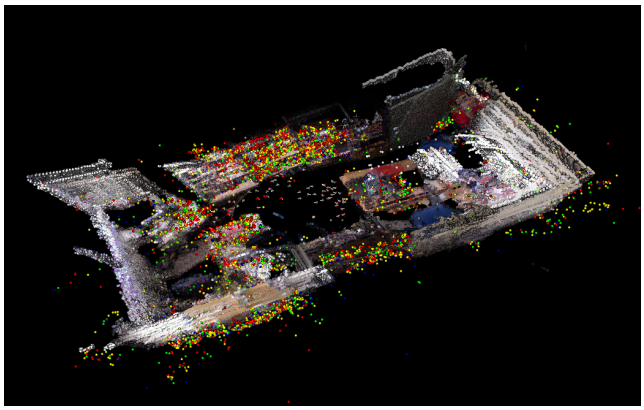


Figure 20: 3D map reconstruction by the two cameras.

We believe that the allocation of the bulky map estimation and management in a Cloud of servers outside the robot sets the basis for the mapping systems to exploit the next step of computing power in order to keep track of Moore's Law. Additionally, it opens the door to a new array of possibilities: 1) the online estimation can be massively parallelized and hence very large maps can be optimized in a short time (like in [39]); also, parallelizing the relocation could boost the number of maps that can be managed in a reasonable time. 2) The developed algorithms provide the basis for the interface of a map database in the Cloud. 3) The map can be improved and enriched offline with expensive computations that cannot be done online, both geometric (e.g., map smoothing assuming a planar environment [40] or free space estimation [9]) and semantic (e.g., recognition of objects [6, 7]). The new geometric or semantic features will be available if a later user relocates itself in a previous map and downloads this map. 4) The massive storage of maps in the Cloud could serve as a training database for learning algorithms to model the commonalities of robotic maps and their variations in the temporal dimension.

8. Acknowledgments

This paper has been partially funded by the European Union Seventh Framework Programme FP7/2007-2013 248942 RoboEarth, the Spanish MICINN DPI2009-07130 and DPI2012-32168 grants and the Aragón regional grant DGA-FSE (grupo T04). The authors are grateful to Marta Salas and Oscar G. Grasa for fruitful discussions.

References

[1] H. Durrant-Whyte, T. Bailey, Simultaneous localisation and mapping (SLAM): Part I the essential algorithms, Robotics and Automation Magazine 13 (2) (2006) 99–110.

[2] K. Goldberg, B. Kehoe, Cloud robotics and automation: A survey of related work, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5.

[3] G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in: Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007.

[4] M. W. Achtelik, S. Weiss, S. Lynen, M. Chli, R. Siegwart, Vision-based MAV Navigation: Implementation Challenges Towards a Usable System in Real-Life Scenarios, in: Workshop on Integration of perception with control and navigation for resource-limited, highly dynamic, autonomous systems, in Robotics: Science and Systems (RSS), 2012.

[5] W. Churchill, P. Newman, Practice makes perfect? managing and leveraging visual experiences for lifelong navigation, in: IEEE International Conference on Robotics and Automation (ICRA2012), 2012.

[6] J. Civera, D. Gálvez-López, L. Riazuelo, J. Tardós, J. Montiel, Towards semantic slam using a monocular camera, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 1277–1284.

[7] M. Salas, J. Montiel, Keyframe based semantic mapping, Tech. rep. (December 2011).

[8] D. Hoiem, A. Efros, M. Hebert, Recovering surface layout from an image, International Journal of Computer Vision 75 (1) (2007) 151–172.

[9] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems, in: Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation, 2010.

[10] B. Russell, A. Torralba, K. Murphy, W. Freeman, Labelme: a database and web-based tool for image annotation, International Journal of Computer Vision 77 (1) (2008) 157–173.

[11] K. Lai, L. Bo, X. Ren, D. Fox, A large-scale hierarchical multi-view rgb-d object dataset, in: IEEE International Conference on Robotics & Automation (ICRA), 2011.

[12] A. Torralba, A. Efros, Unbiased look at dataset bias, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011, pp. 1521–1528.

[13] A. Mourikis, S. Roumeliotis, Predicting the performance of cooperative simultaneous localization and mapping (c-slam), The International Journal of Robotics Research 25 (12) (2006) 1273–1286.

[14] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, S. Teller, Multiple relative pose graphs for robust cooperative mapping, in: Robotics and Automation (ICRA), 2010 IEEE International Conference on, 2010, pp. 3185–3192.

[15] H. Strasdat, J. Montiel, A. Davison, Real-time Monocular SLAM: Why Filter?, in: IEEE International Conference on Robotics and Automation (ICRA), 2010.

[16] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[17] E. Guizzo, Robots with their heads in the clouds, IEEE Spectrum 48 (3) (2011) 16–18.

[18] S. Remy, M. Blake, Distributed service-oriented robotics, IEEE Internet Computing 15 (2) (2011) 70–74.

[19] M. Waibel, M. Beetz, R. D'Andrea, R. Janssen, M. Tenorth, J. Civera, J. Elfring, D. Gálvez-López, K. Haussermann, J. Montiel, A. Perzylo, B. Schiele, O. Zweigle, R. van de Molengraft, Roboearth - a world wide web for robots, IEEE Robotics and Automation Magazine 18 (2) (2011) 69–82.

[20] H. Bistry, J. Zhang, A cloud computing approach to complex robot vision tasks using smart camera systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010, pp. 3195 –3200.

[21] J. Rogers, A. Trevor, C. Nieto-Granda, H. Christensen, Simultaneous localization and mapping with learned object recognition and semantic data association, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 1264–1270.

[22] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. Kumar, K. D. Meng, G. W. Kit, Davinci: A cloud computing framework for service robots, in: IEEE International Conference on Robotics and Automation (ICRA), 2010, pp. 3084 –3089.

[23] A. Wendel, M. Maurer, G. Graber, T. Pock, H. Bischof, Dense reconstruction on-the-fly, in: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 1450–1457.

[24] R. O. Castle, G. Klein, D. W. Murray, Video-rate localization in multiple maps for wearable augmented reality, in: 12th IEEE International Symposium on Wearable Computers, 2008, pp. 15–22.

[25] R. Castle, G. Klein, D. Murray, Wide-area augmented reality using camera tracking and mapping in multiple regions, Computer Vision and Image Understanding.

[26] G. Klein, D. Murray, Improving the Agility of Keyframe-Based SLAM, in: Proceedings of the 10th European Conference on Computer Vision: Part II, Springer, 2008, pp. 802–815.

[27] B. Triggs, P. McLauchlan, R. Hartley, A. Fitzgibbon, Bundle adjustment – A modern synthesis, in: Vision Algorithms: Theory and Practice, LNCS, Springer Verlag, 2000, pp. 298–375.

[28] J. Civera, A. Davison, J. Montiel, Interacting multiple model monocular SLAM, in: IEEE International Conference on Robotics and Automation (ICRA), 2008, pp. 3704–3709.

[29] S. Gauglitz, C. Sweeney, J. Ventura, M. Turk, T. Hollerer, Live tracking and mapping from both general and rotation-only camera motion, in: Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on, IEEE, 2012, pp. 13–22.

[30] E. Rosten, T. Drummond, Machine learning for high-speed corner detection, European Conference on Computer Vision (ECCV) (2006) 430–443.

[31] A. Torralba, R. Fergus, W. T. Freeman, 80 million tiny images: A large data set for nonparametric object and scene recognition, Pattern Analysis and Machine Intelligence, IEEE Transactions on 30 (11) (2008) 1958–1970.

[32] M. Milford, Vision-based place recognition: how low can you go?, The International Journal of Robotics Research 32 (7) (2013) 766–789.

[33] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 1998, pp. 604–613.

[34] G. D. Tipaldi, L. Spinello, W. Burgard, Geometrical flirt phrases for large scale place recognition in 2d range data, in: Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE, 2013, pp. 2693–2698.

[35] F. Moreno-Noguer, V. Lepetit, P. Fua, Accurate non-iterative o(n) solution to the pnp problem, Computer Vision, IEEE International Conference on 0 (2007) 1–8. doi:http://doi.ieeecomputersociety.org/10.1109/ICCV.2007.4409116.

[36] L. A. Clemente, A. J. Davison, I. D. Reid, J. Neira, J. D. Tardos, Mapping large loops with a single hand-held camera, in: Robotics: Science and Systems, 2007.

[37] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, Ros: an open-source robot operating system, in: ICRA Workshop on Open Source Software, 2009.

[38] G. Klein, D. Murray, Parallel tracking and mapping on a camera phone, in: 8th IEEE International Symposium on Mixed and Augmented Reality, 2009, pp. 83–86.

[39] Y. Furukawa, B. Curless, S. Seitz, R. Szeliski, Towards internet-scale multi-view stereo, in: IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 1434–1441.

[40] D. Gallup, J. Frahm, M. Pollefeys, Piecewise planar and non-planar stereo for urban scene reconstruction, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2010, pp. 1418–1425.