Philipp Fleck

Multiuser SLAM

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Clemens Arth
Institute for Computer Graphics and Vision

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg
Institute for Computer Graphics and Vision

Graz, Austria, April. 2015

Abstract

We present a method to combine multiple local SLAM maps into combined maps in a client-server system. The server takes care of all clients and tries to detect overlapping regions among keyframes committed by clients. The system supports different clients with different levels of complexity, such as a thin client, which is used for image acquisitions, or an autonomous SLAM client, which generates its own local map. If clients move, the combined map is refreshed to keep pace with the client's local map. Beyond the combination of client maps, the server system can update clients to improve their local system using keyframes and poses. Allowing clients to operate in the same context, will serve as a base for future AR applications. In particular, multiple clients commit their keyframes and the server generates a per-client reconstruction, as well a combined map. Afterwards, the clients receive updates in form of new keyframes and poses, to improve and enlarge their local system.

Keywords. SLAM, Multiuser-SLAM, AR, SfM, multiclient, server reconstruction, map combination, map expansion

Kurzfassung

In dieser Arbeit präsentieren wir ein Client-Server System zur Erstellung von 3D Karten. Benutzer können dabei ein SLAM-System verwenden (mobiler Computer, Smartphone), um eine lokale 3D Karte zu erzeugen. Während der Erstellung, können Bilder der beobachteten Szene und weitere Information wie Kamera Posen zum Server übertragen werden. Das Server-System erstellt für jeden Benutzer eine 3D Karte und versucht die 3D Karten mehrerer Benutzer zu größeren Karten zu kombinieren. Ist es möglich zwei oder mehr Benutzer zu kombinieren, werden alle betroffen Benutzer vom Server aktualisiert, um ihr lokales System zu verbessern oder ihre lokale 3D Karte zu vergrößern. Lokal verwendete Systeme können von sehr einfach (kein Tracking, es werden nur Bilder übertragen) bis hin zu sehr komplex (lokales SLAM System, dass auch Informationen vom Server-System beziehen kann) reichen. Haben nun mehrere Benutzer eine gemeinsame 3D Karte erstellt, kann diese als Grundlage für verschiedene AR Applikationen dienen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented master's thesis dissertation.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Acknowledgments

Writing this thesis was a big effort and of coarse a challenging task. Therefore, i thank my supervisors Dr. Clemens Arth and Prof. Dr. Dieter Schmalstieg from the Institue for Computer Graphics and Vision in Graz for the great guidance and support over the last eight months. Especially Clemens helped me with long discussions and pointed me in the right direction. Additionally i would like to thank all people from the institute for some nice discussions. Especially Christian for his great input.

Further, big thanks to my parents, which are always great support and they take into account for my almost strict time schedule. Special thanks to Vanessa, my girlfriend since eight years, who always helps me with everything, motivates me if i leak on motivation and is always on my side.

Also my friends deserve thanks, which never comply on my strict time schedule and always are good company.

I would also like to thank my former boss Hannes at Magna Steyr, who allowed flexible work times and was tolerant to exams and the related (study)vacation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

Creating a 3D reconstruction of the environment through the use of Computer Vision has become mature recently. Those 3D models can be used for many applications in architecture, engineering, navigation and robotics. The main classes of algorithms are those that are (1) named as SfM, while typically run offline and process huge amounts of visual and sensory data, and those (2) which target real-time applications using more lightweight algorithms and restrict themselves to certain types of input data.

The first class, *Structure from Motion*, can use ordered or unordered sets of images taken by one or multiple cameras. Vision based algorithms, retrieve 3D information from image correspondences, which are often calculated using natural feature detectors like SIFT [39]. If uncalibrated cameras are used, the calibration is based on the detected natural features. Inaccurate camera calibrations lead to inaccurate reconstructions. SfM is similar to the stereo reconstruction problem. Usually, the 3D reconstruction is a sparse point-cloud including the camera trajectories. Current approaches mainly operate on outdoor data towards city-scale reconstructions.

The second class, *Simultaneous Localization and Mapping*, creates a map of the environment and localizes the current view, while using observations of 3D points, within the created map. The aim is to provide a precise position and orientation to the operating application, which could feed navigation-systems or any kind of AR-applications. Such systems have to be incremental to enlarge the reconstruction, when moving towards unobserved areas. Beyond visual algorithms, sensors like GPS (*Global Positioning System*) or LiDAR (*Laser detection and Ranging*) can be taken into account. Due to the lightweight nature of these algorithms, algorithmic parts have to be well chosen, to deliver results as good as necessary, when operated in real-time (e.g. replacing SIFT with a more lightweight feature detector).

Although the clean separation of approaches into these classes has been blurred recently through the advent of more powerful hardware in the desktop and mobile section,

from the algorithmic point of view, distinctive properties and features are still visible. For the applications in AR, where real-time applicability is constantly a key issue, the second class has been accepted. SLAM approaches enable users to interact with their environment at a very intuitive and extensive way, allowing for playing games or interacting in architecture. However, SLAM systems typically operate in the single user domain, and adopting operability and applications to the multi-user domain is not straightforward.

When moving applications into the multi-user domain, certain questions become relevant.

How will multiple users communicate?

Can 3D information from one users space be transferred into another users space?

How can different users interact?

Is there a need for a common 3D space?

We are going to tackle some of these questions with our approach. To generate a common foundation for communication and data exchange, it is necessary to operate within the same 3D space. User can receive information already transformed into the local space or users are transformed into the shared space. Peer-to-Peer as well as client-server approaches look promising. We will show, how to solve some of the named problems, when using a client-server system, where users can register and collaboratively create a 3D reconstruction. This system will also serve as a common foundation for all users.

1.2 Method

In this work, we present a method for scalable collaborative SLAM. With a key aspect on combining multiple SLAM maps into bigger ones. Our approach uses a server-client architecture, without strong dependencies between a server and its clients, which allows the usage of clients with different levels of complexity. Thin clients may take high resolution pictures (significantly higher as in current SLAM systems) and commit those, without further processing for more detailed reconstructions. Fat clients may operate a local SLAM system and refine their own measurements, using information provided by the server. The server performs SfM per client and tries to combine clients, by detecting overlapping regions in images of the observed scene. Our proposed method detects overlaps among images, using image matching with natural features. To combine two maps, we search for correspondences between image features and triangulated points, or between triangulated points of two related maps. Based on this information, we calculate a 3D transformation to align both maps. We can distinguish between the following cases of possible map combinations: (i) combining two clients to get a merged map. (ii) combining a client with a merged map to expand this map. (iii) combining two merged maps to get a bigger merged map.

Therefore multiple clients may generate a combined map and update their local information, by the information generated by other clients. In detail, the server system holds an individual data-queue per client, which is updated if the combined map changes (e.g. a new client is added or the map of a participating client grew). The provided data, like keyframes and camera positions and orientations, is individually prepared for each client. Clients can then expand their local maps or improve their tracking performance based on the received data. For instance, when adding received keyframes to the local SLAM system, measurements like observations of certain 3D points may be added.

Operating within the same coordinate system, allows Augmented Reality (AR) applications to share similar information among different clients. The motivation is to generate merged maps, enlarge them when clients are discovering new areas of the environment, and let clients know, what is beyond their current view. This enlarges the range of possible applications considerably over the single-user use case. Imagine a city could be stepwise reconstructed. Separate parts would be connected by participating clients. New clients (e.g. tourists) could navigate without the need of creating a map, and could possibly use annotations provided by other clients. Further, the server information could be updated over time and the collaborative reconstruction could improve or stay up to date.

Another application could be a simple card game. Each player would view the same content from his perspective. Annotations or other information would be shared among the players, and each player could interact with the scene, affecting the other players. The problem of how to represent information among different clients would disappear, because each client would participate to the same map. An obvious application is navigation. Self-driving vehicles could explore different parts of the environment, which would then be combined to a global representation. Subsequently passing vehicles would then already know the area and could navigate with less danger.

We provide a scalable SLAM-System, with knowledge from all clients improving the individual client-performance, in terms of localization and tracking, by allowing clients to actualize and expand their local information with available updates. Knowledge about clients operating in the same environment is achieved by detecting, which clients observe the same scene, and combining their maps and poses. The server system stores generated maps in a pool and lets clients participate if they reside in the same environment as a certain map of the pool. The advantages of our system, spread from supporting a big variety of clients up to improving their performance and serving a common coordinate system for data exchange. Extending existing clients with our our server system, causes negligible effects on real-time performance and allows lightweight implementations. Further, clients can handle incoming poses and keyframes like local ones, which also reduces additional computational effort. Additionally, our system does not restrict to certain client implementations and allows clients to decide on their own, when and how much data to pull from the server.

1.3 Outline

The rest of this thesis is structured as follows. Chapter 2 explains the mathematical foundation and mathematical notations of this work, defining the terms camera, pose and calibration. Subsequently the epipolar geometry of a stereo camera setup and how the essential and the fundamental matrices are calculated is described in the latter points of this section. We present algorithms, which are used to calculate poses from point correspondences. Finally an overview of bundle adjustment, and a description of $SO3$ and $SE3$ Lie-groups.

Chapter 3 is divided into two parts. First we give an overview over relevant work about structure from motion, followed by an description of relevant work in the area of simultaneous tracking and mapping.

Chapter 4 presents our work and gives a closer look of individual parts of our system. First, we give a rough overview of the SfM pipeline used in our work and illustrate example reconstructions. Second, the proposed server system is explained, going into detail on how to perform various operations, like the detection of overlapping images or the combination of two maps. Furthermore, differences between the *3 point pose and scale estimation* algorithm and the *Horn alignment* algorithm are shown, as well the chosen client-server communication approach. In the remainder of the section, we present different clients, like a thin client with no SLAM and an autonomous SLAM client. Our three operational modes for SLAM clients are explained as well. Finally we report details about the current implementation of the server-system and the client implementation.

Chapter 5 shows details about the executed experiments, where each part describes one test environment in detail. First, we describe the *Carpet* scene, which is used to test the imeplemented merge algorithm for various clients. Then, we describe the test environments *Cardgame* and *Office*. Cardgame is used to test different client modes, while Office is a general experiment. Finally, we give additional thoughts on the executed experiments and show some real application examples.

Chapter 6 summarizes the proposed approach and gives an outlook to ongoing work with further examples of possible applications and concluding remarks.

2 Mathematical foundation

This section describes basic mandatory algorithms and related mathematical notations, which are used in our work. Matrices are denoted as bold capital letters like $\mathbf{R}$ and column vectors are written as bold small letters like $\mathbf{x}$. The italic capital letter I is denoted to images and we refer to scalars as small italic letters like s. The capital letter X denotes a 3D point and the small letter x denotes a 2D point.

2.1 Camera

A pose defines the rotation and translation of the camera with respect to a given coordinate system. The pose $\mathbf{P}$, a 3×4 matrix, can transform 3D points of a world coordinate system into the camera coordinate system. $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{c}$ (3×1) defines the camera center. $\mathbf{R}$ and $\mathbf{t}$ form $\mathbf{P}$, as shown in equation 2.2. The camera calibration matrix $\mathbf{K}$, which captures the intrinsic parameters of the camera, is used to project 3D points from the camera coordinate system onto the image plane of the camera. f_x and f_y describe the focal length in x and y direction and p_x and p_y are the position of the principal point in the image plane. For more details about camera representation and calibration, the interested reader is referred to [26]. A keyframe defines a single image or frame, which is taken at a certain point of time, in a subsequent chain of frames used for reconstruction. Due to the orthogonality of the rotation matrix, we can invert the pose using equation 2.4. The inverse pose $\mathbf{P}^{-1}$ is used to transfer a point from the camera coordinate system into the world coordinate system. Equation 2.5 shows how to project a 3D point X onto the image plane of the camera, where w is used to normalize x.

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\mathbf{P} = [\mathbf{R}|\mathbf{t}] \tag{2.2}$$

$$\mathbf{t} = -\mathbf{R} * \mathbf{c} \tag{2.3}$$

$$\mathbf{P}^{-1} = [\mathbf{R}^T | -\mathbf{R}^T * \mathbf{t}] \tag{2.4}$$

$$\begin{pmatrix} \mathrm{x} \\ w \end{pmatrix} = \mathbf{K} * \mathbf{P} * \begin{pmatrix} \mathrm{X} \\ 1 \end{pmatrix} \tag{2.5}$$

2.2 Epipolar Geometry

The epipolar geometry describes the relations of a stereo image pair I_1, I_2 of the same scene. Further, the epipolar lines l and l' are the projection of one epipole into the other image and the corresponding epipoles e and e' are the intersection of the baseline with the image plane, as shown in figure 2.1. The Fundamental Matrix $\mathbf{F}$, is used when uncalibrated cameras are used and maps a point of one image to the corresponding epipolar line in the other image $l' = \mathbf{F}\mathrm{x}$, which follows from $\mathrm{x}'^T\mathbf{F}\mathrm{x} = 0$. The Essential Matrix $\mathbf{E}$ is the calibrated version of $\mathbf{F}$ and defined by $\mathbf{E} = \mathbf{K}^T\mathbf{F}\mathbf{K}$, where $\mathbf{K}$ is the camera calibration (see equation 2.1). In the case of different cameras, both camera calibrations have to be taken into account. Epipoles are defined by the intersection of the baseline and the image plane.

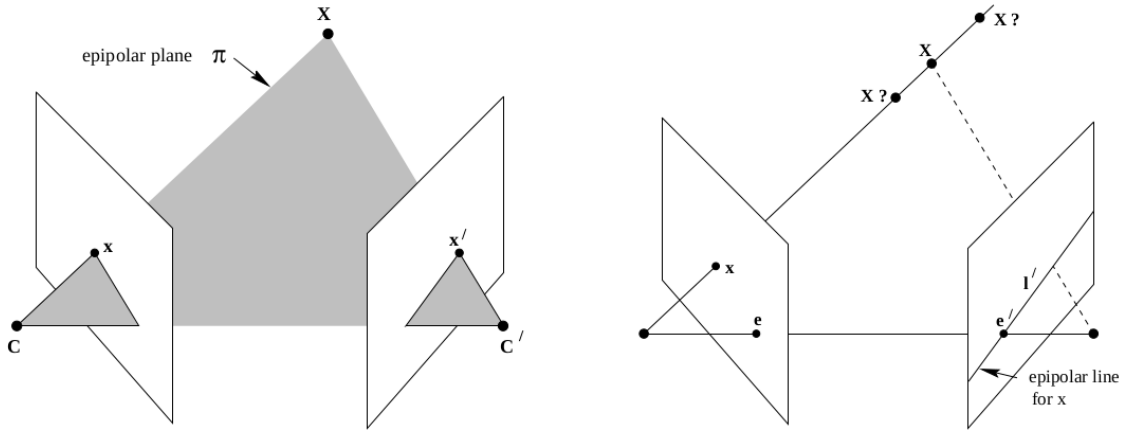


Figure 2.1: Epipolar geometry showing image correspondeces x and x', the epipoles e and e' with corresponding epipolar line l and l'. Image from [26].

2.3 Stereo reconstruction

The epipolar geometry is used to verify image point correspondences and to discard outliers. Points are triangulated by solving equation 2.6 for $\mathbf{X}$ using the singular value

decomposition (*SVD*, see [25]). X defines a 3D point, **P** and **K** define the pose and the camera calibration, and x_i and x_i' are the corresponding image points of each image.

$$x_i' \Leftrightarrow x_i, \qquad \left.\begin{array}{c} x_i = \mathbf{K}\mathbf{P}X_i \\ x_i' = \mathbf{K}'\mathbf{P}'X_i \end{array}\right\} \Rightarrow \mathbf{A}\mathbf{X} = 0, \qquad \mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{2.6}$$

For each further keyframe, one may get more observations $(x_i, x_i', x_i'', \dots)$ for a certain 3D point (X). It is very likely to happen, that results of solving equation 2.6 may vary with increasing number oft points. To compensate for this behaviour, a so called bundle adjustment step is needed to optimize 3D point positions and camera positions and orientations. A more detailed explanation of the bundle adjustment follows in section 2.7.

2.4 5-Point-Pose

The 5PP algorithm estimates the relative rotation and translation of a camera with respect to the other given one, using five or more 2D-2D point correspondences. Over the last years many solutions have appeared to this problem. More recent approaches are shown by Nistér [42], Stewenius *et al.* [52] and a fast iterative approach is shown by Hedborg and Felsberg [27]. Figure 2.2 shows five point correspondences from two images with their 3D points. The idea is to solve equation 2.7, where x' and x are corresponding 2D points, K_1 and K_2 define the camera calibration matrices and **E** is the essential matrix.

$$x'(\mathbf{K}_2^{-1})^T\mathbf{E}\mathbf{K}_1^{-1}x = 0 \tag{2.7}$$

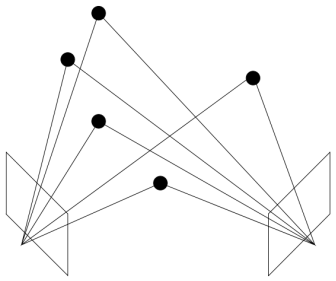


Figure 2.2: Five point pose estimation. Image from [52].

The following five steps show the basic idea of Stewenius approach.

(i) The linear equations of the epipolar constraint are used to parametrise the essential matrix, with three unknows.

(ii) Rank and trace constraints are used to build ten polynomial equations in the three unknowns.

(iii) the Gröbner basis is computed.

(iv) A 10×10 action matrix is computed.

(v) The eigenvectors of the action matrix are computed. Back-substitute to (i) to obtain the solution for the essential matrix.

2.5 3-Point-Pose

3PP is used to estimate the camera pose relative to a given set of 3D points. Therefore, at least three 2D-3D point correspondences are needed, which are used to form three arbitrary rays to determine the pose. Three point correspondences are not sufficient for a definitely solution and deliver up to four different solutions. At minimum four point correspondences are needed for a unique result. A possible algorithm, as shown by Nistér *et al.* in [43] is performed in six steps, which includes line up of rays and points in their canonical position. Followed by the calculation of plane and polynomial coefficients. Extracting the roots of the polynomial and back-substituting them, will deliver the pose of the camera in the given 3D point set.

The *POSIT* algorithm, introduce by DeMenthon and Davis [16], takes 2D-3D correspondences and camera parameters and returns rotation and translation of the pose. Figure 2.3 shows the orthographic projection as used in the algorithm.

When using four points it boils down to equation 2.8, where s is the scale, $\mathbf{K}$ and $\mathbf{P}$ are the camera calibration and the camera pose, and x and X are 2D-3D correspondeces, as shown by Bujnak *et al.* [11].

$$s * \mathrm{x} = \mathbf{KP}\mathrm{X} \tag{2.8}$$

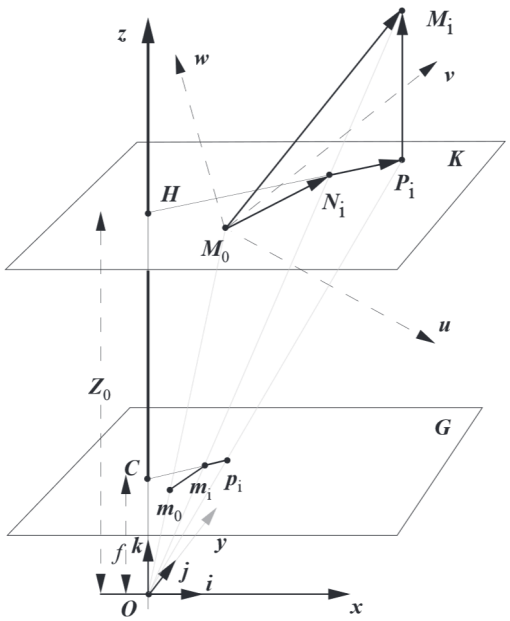


Figure 2.3: Three point pose estimation. Scaled orthographic projection as used in the POSIT algorithm. Image from [16].

2.6 Horn alignment

Horn Alignment uses 3D-3D point correspondences to calculate the transformation between two sets of points, including scale e.g. 3D points of a point cloud A corresponding to 3D points of point cloud B. There are at least four point correspondences required, to determine the rotation matrix $\mathbf{R}$, the translation vector $\mathbf{t}$ and the scaling parameter s. In particular, the algorithm initially calculates the centroids of each point-set, using equation 2.9. The scale can then be directly calculated from $\mathbf{c_A}$ and $\mathbf{c_B}$, where $||\cdot||$ defines the euclidean distance (see equation 2.10). $\mathbf{R}$ is determined by calculating the singular value decomposition on the outer product of all correspondences, and afterwards multiplying with the calculated SVD components, as shown in equation 2.13. Note that, before calculating the SVD, one point set has to be scaled by s or s^{-1}, according on how s is calculated in equation 2.10. The translational vector can be calculated by using the centroids following equation 2.14. A closed form solution of the algorithm is shown by Horn [30].

$$\mathbf{c_A} = \frac{1}{N} * \sum_{i=1}^{N} \mathrm{X_{Ai}} \ , \quad \mathbf{c_B} = \frac{1}{N} * \sum_{i=1}^{N} \mathrm{X_{Bi}} \tag{2.9}$$

$$s = \frac{1}{N} * \sum_{i}^{N} \frac{||\mathrm{X_{Ai}} - \mathbf{c_A}||}{||\mathrm{X_{Bi}} - \mathbf{c_B}||} \tag{2.10}$$

$$\mathbf{L} = \sum_{i=1}^{N} \mathrm{X_{Ai}} \times s * \mathrm{X_{Bi}} \tag{2.11}$$

$$\mathbf{UDV}^T = SVD(\mathbf{L}) \tag{2.12}$$

$$\mathbf{R} = \mathbf{LVDV}^T \tag{2.13}$$

$$\mathbf{t} = \mathbf{c_A} - \mathbf{R} * \mathbf{c_B} \tag{2.14}$$

The resulting rotation matrix $\mathbf{R}$, the translation vector $\mathbf{t}$ and the scaling s are a transformation between two 3D spaces. Using this we can easily align two pointclouds. 3D points can be transformed using equation 2.16. Poses can be transformed using equation 2.17. In equation 2.15 $\mathbf{0}$ denotes a 3D vector of zeros.

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} * s \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.15}$$

$$\mathrm{X}' = s * (\mathbf{R} * \mathrm{X} + \mathbf{t}) \tag{2.16}$$

$$\mathbf{P}' = \mathbf{P} * \mathbf{M}^{-1} \tag{2.17}$$

2.7 Bundle adjustment

Bundle adjustment is a non-linear optimization technique, which is often used in 3D reconstructions or SLAM systems to improve point and pose estimates. Incorporating all calculations at once, Nistér and Engels give a good description in their work [21], on how to refine data, achieved by a static camera and an object placed on a turning table in front. Without bundle adjustment, the received circular positioned poses would be on a deformed circle, caused by incremental errors and computational inaccuracy. Applying bundle adjustment after each added frame, they could achieve poses, placed on a perfect circle around the object (see figure 2.4). However, key for good bundle adjustment results, is the initialization as bundle adjustment can not compensate for poor initializations. Further, it improves the overall performance, by improving each initialization of a new pose, by correcting all previous ones. The basic idea is to minimize the re-projection error[1] over all known frames. Further approaches to bundle adjustment are shown in [3, 38]. An actively developed project is the *Ceres Solver* [2], which includes a big selection of different solvers and is rather fast.

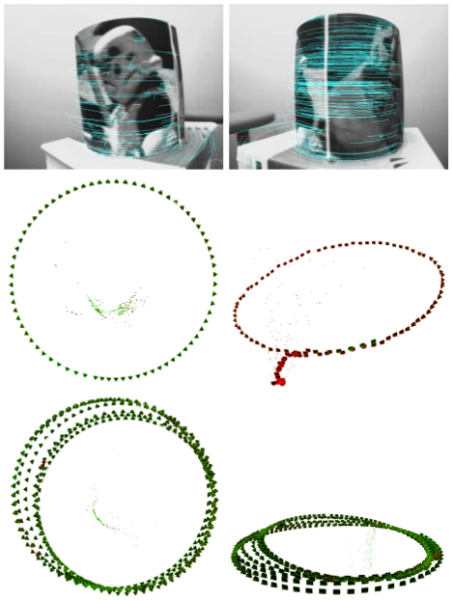


Figure 2.4: Bundle adjustment. The top row shows the object placed on a turning table. The middle and last row show camera poses with and without bundle adjustment. Image from [21].

[1] Difference between measurement of a point and the projection of its 3D-point onto the image plane of a certain camera.

2.8 Lie-Algebra

Because this is a very big field in mathematics, which is used for operations among geomtrical bodies, we concentrate on the specific groups $SO3$ and $SE3$. $SO3$ describes the group of orthogonal rotation matrices and allows to represent a 3×3 rotation matrix as an unique 3×1 vector. $SE3$ expands $SO3$ while adding a translation vector to the rotation. Regarding to SLAM and SfM systems we can use $SE3$ to represent a pose. One advantage is to reduces the amount of stored data. Further, we can lossless transform back and forth between the notations. An advantage over euler angles is, that rotation is unique and does not require a e.g. XYZ notation like euler angles. More about Lie-Algebra can be found in Knapp's work [36] and [44, 47].

$$\mathbf{P} = [\mathbf{R}|\mathbf{t}] \tag{2.18}$$

$$SE3(\mathbf{P}) = \begin{pmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{pmatrix} \tag{2.19}$$

3 Related Work

In the following we will review related work from the area of SfM and SLAM, and will discuss closely related approaches on solving the multi-user problem in detail.

3.1 Structure from motion

Over the last years different approaches to SfM have been established. We list some of them, which appear relevant to our work. Images used as input for SfM systems may have various sources, including big image databases with unsorted images like Flickr[1], (sequential) images retrieved by UAVs (*Unmaned Areal Vehicle*) or any other image acquisition device. Such devices range from smartphones, optical cameras to cameras with integrated depth sensors. Reconstructions reach from small scale like an office to world-sized as it is done in the Google StreetView[2] project. Since it was a far way to UAVs, early works show already high sophisticated algorithms for scene reconstruction. Amstrong *et al.* show in their work a two step process on how to generate euclidean reconstructions from uncalibrated images [6]. In particular, they recover structure up to affine ambiguity from two views and use one or more additional views to determine the intrinsic parameters of the camera. Further, Baillard *et al.* [7] show how to automatically reconstruct piecewise planar models from multiple views. They utilize inter-image homographies to validate and estimate best planes for the reconstruction. 3D model acquisition from extended image sequences is performed by Beardsley *et al.* [9], where token based matches across an image triplet are used to compute a 3D structure.

Snavely *et al.* show, how to browse thousands of images and how to visualize them in a sparse 3D model, using an incremental approach [50]. In detail, they add one image at the time and optimize with bundle adjustment, as shown later in [51]. Further, their system performes well on images of major landmarks, but fails to reconstruct from images, which are not connected to the initial reconstruction, resulting in the usage of a smaller

[1] Flickr, `https://www.flickr.com/`

[2] `https://www.google.com/maps/views/streetview`

subset of large photo collections. Using large unsorted datasets, leads to slowdowns in the later reconstruction, therefore they consider to somehow pre-order images. Based on this work Snavely created the bundler[3] software, which can handle unordered image collections, using the SBA (*Sparse Bundle Adjustment*) introduced by Lourakis and Argyros [38]. SBA is a software package, that utilizes the sparsity of the jacobians in bundle adjustment to gain computational savings. The bundle software has become the standard reconstruction tool for the scientific community to compare novel SfM approaches to.

Further, Agarwal *et al.* show in *Building Rome in a Day* [1] how to remove sequential bottlenecks and how to maximize parallelism, while processing 150K images on 500 compute cores, allowing reconstructions of whole cities. Their approached system scales with the problem and also with the available computational resources, which becomes important, when trying to process datasets with millions of images. Due to the analysis of online datasets, they discovered that there is a high probability of matching images taken by single users. This fact is utilized in the proposed skeletal algorithm. One drawback is, the challenging problem of starting with finished reconstructions, and enlarging them with new images from online databases.

Irschara *et al.* use calibrated cameras for large-scale reconstruction, with the goal to serve a wiki-based system for information retrieval [31]. By applying the wiki principle (collaborative intelligence, articles (models) get better if more users participate), they avoid using online databases like Flickr as used in other works, but they tend to use user based participations, to generate a photo-realistic model of an urban environment. Generated models can be maintained and gradually refined by adding new images.

Pollefeys *et al.* adapt current state of the art algorithms towards GPU, to achieve rael-time performance demonstrated on urban video sequences [4, 45]. In particular, they operate on video streams with acquired GPS information. Because of the large dynamic range of outdoor environments, the system has to estimate the global camera gain for future frames, and has to compensate for it. The focus of the system is on redundancy between frames (redundant information may be seen across tens of frames). Based on this, a 2D tracker tracks features over frames for possible 3D points before reconstruction. The final result are ground-based and dense geo-registered models.

A slightly different approach is taken by Strecha *et al.* in [54], when trying to create a connected model, by avoiding the reconstruction of loose connected sights, under usage of the provided image meta-data. Bundle adjustment is applied on smaller clusters to avoid high memory footprints, instead of operating on whole large-scale maps. If we consider our system in terms of combining clients, bundle adjustment on smaller clusters might be a key feature when it comes to save computation time and to lower the memory usage. Further they are able to update their model when new images arrive.

Vergauwen and Van Gool show, how to create a webbased 3D reconstruction application operating with self calibrating cameras [60]. A global image comparison algorithm

[3]Bundler, `http://www.cs.cornell.edu/~snavely/bundler/`

based on normalized cross correlation, is used to rapidly identify pairs of images, which are very likely to match and be used for 3D reconstruction. Dense matching is used to assure that enough 3D points can be calculated for a high quality 3D reconstruction. Users are able to upload their images and to receive a 3D model using the provided application. Further they show how to realise a queueing system, operating with a compute cluster. The proposed system was in beta testing at the time of submission (2006) and was used by several users to create 3D models from cultural heritage related images.

Hoppe *et al.* shows a scalable and computationally more efficient approach of SfM [32]. In particular, they propose a view selection strategy, which is based on GPS/IMU (Inertial Measuring Units), to reduces matching efforts. Additionally a fast and scalable reconstruction approach, based on global rotation registration and robust bundle adjustment, is shown. In [12], Hoppe *et al.* show an online SfM approach, with an surface extraction for visualization.

The number of approaches described is far from exhaustive. However, there is a clear trend towards real-time reconstruction recently, which is mainly driven by the increasing availability of compute clusters and low-cost GPUs. In particular, our mean SfM server part is based on the work of Hoppe and does currently not use positioning data. Further, Vergauwen and Van Gool point to a real-time queueing-system and how to parallelize reconstruction, which is also considered in our work. Strechas approach, to apply bundle adjustment on clusters of large-scale maps to void big memory footprints and higher computational efforts, is also considered an optional addition to our system.

3.2 SLAM systems

In this section we point at existing systems, like *PTAM*, *PTAMM* or C^2TAM, which may perform in realtime or even in a cloud. PTAM is one of the first applications, allowing real-time operations, PTAMM added a multi-map system. C^2TAM moved the mapping part to the server and the tracking part to the clients.

More than twenty years ago, navigational problems with robots in known and unknown environments where first investigated. Since then, various solutions have been proposed. Smith *et al.* show how to estimate spatial relationships in [49], and Chatila *et al.* show solutions to consistent modelling the environment for mobile robots. Later, Dissanayake proposed a solution to the SLAM problem in [18] and Montemerlo presented FastSLAM, as operating SLAM system, which also tackles the loop closure problem [40]. Loop closure is the problem of detecting revisited areas in a certain map, to avoid recreating already created parts of the map. Solutions to this problem have been shown by Ho in [28] and Angeli in [5]. Davison proposed a bayesian based SLAM system in [14], operating with one camera and robust (re)localization. A comparison between filter based and keyframe based SLAM systems, is show by Davison *et al.* in [53], with the conclusion that keyframe based systems with bundle adjustment outperform filter based ones, in terms of accuracy

per unit of computing time.

Davison *et al.* introduced MonoSLAM, which is a probabilistic approach to the SLAM problem, showing that realtime (drift-free) performance is possible, with pure vision of a single uncontrolled camera [15]. The central part of the system is a sparse map of natural landmarks of the environment, which is created at runtime, using a probabilistic framework. Further, a motion-model for smooth camera movement is applied and solutions for monocular feature initialization and feature orientation estimation are presented.

Klein and Murray presented PTAM, achieving real-time performance by performing tracking and mapping in parallel [34]. Key aspect is, that while using a single camera mapping operations operate on keyframes and no more on each frame, which may often contain redundant information e.g. still standing versus moving camera. The computation of a single keyframe has to be finished with the next keyframe, and no longer with the next frame, allowing instant point triangulations of new features. Mandatory is the dense initialization of the system, otherwise further tracking may easily fail. In [35], PTAM is ported to mobile phones and achieves real-time performance, including 3D objects directly rendered into the tracked scene.

Wendel *et al.* use the PTAM tracker on UAVs and the mapper on the server [61]. Their aerial vehicle sends keyframes and poses to the server for reconstruction. The reconstructed scene is visualized on an interactive visualization client, a tablet.

Tan *et al.* proposed a method to handle dynamic environments,by updating stored keyframes and projecting features among keyframes, as shown in [57]. They introduce SIFT with keyframe representation and allow to remove invalid 3D points and keyframes for occluded objects. Further, their system is more reliable for fast camera movements and can handle dynamic environments including objects entering and leaving the scene in realtime.

Newcombe *et al.* [41] show with DTAM, how to perform SLAM without features, by estimating depth maps from selected keyframes and improving the estimates with the underlying video stream. In particular hundreds of images are used to minimize a global spatially regularised energy functional. Therefore, detailed textured depth maps can be estimated for selected keyframes, which leads to a surface patchwork with many vertices. The system is limited to constant brightness and can not hold the performance at dynamic real world illumination.

RGB-D SLAM uses a low-cost camera, with integrated depth sensor, such as Microsoft Kinect, to gain depth information of certain points, as shown by Endres *et al.* [19]. They localize extracted visual keypoints in 3D within the measured depth-maps. To make the system more robust (e.g. highly changing scenes), a quality measurement is introduced, which is used to reject inaccurate measurements.

Izadi *et al.* proposed KinectFusion [33], a system to create 3D reconstructions in real-time. 3D points are tracked using only the depth information, retrieved by the Microsoft Kinect sensor. Additionally, they present novel GPU techniques to allow real-time AR applications.

Engel *et al.* [20] take a different approach, to current state-of-the-art algorithms. Instead of using feature points, they use the image intensity to calculate image alignments, resulting in precise pose estimates. Their system, LSD-SLAM , is capable to operate in large-scale and complex environments in real-time.

3.3 Multi User SLAM/SfM

The following section describes approaches towards the multi-user domain, which are relevant for our work. Castle *et al.* show in [13] with PTAMM, how to handle multiple maps and multiple cameras, using PTAM as foundation. They describe how to localize within multiple maps and how to generate them. Further, it is necessary to allow switching between maps when the user leaves the known area, if maps are not combined and are held as distinctive ones. When tracking is lost, a camera close to the current position is selected and is tried to be re-localised among the stored maps. A drawback of the system is the large memory footprint when growing, which strongly limits scalability.

Riazuelo *et al.* show with their cloud-based framework [37] different approach to PTAM and an evolution of PTAMM. C^2TAM introduces a server-client architecture, where they move the mapping task to the server, and they let clients perform the tracking task. Clients send keyframes to the server for map generation and receive the created map. After each server side bundle adjustment iteration, clients are updated with new map information. The server supports operations to fuse maps. If a client gets lost and the local relocation fails, it can receive filtered keyframes of various maps for localization and, if successful, retrieve the whole related map. Growing maps rapidly increases the amount of transferred data, which is a major bottleneck when using clients with limited bandwidths.

Zou *et al.* show with CoSLAM in [62], how to use multiple cameras to collaborative create a map and to handle dynamic objects. Dynamic objects are tracked among different cameras, and their corresponding 3D points are also marked in the map. The created map can distinct between various types of 3D points, like static points or dynamic points. Further, cameras can be split and merged into groups, based on view dependence, according to their overlap.

Ventura *et al.* combine in their work [59] a server localization system with a local SLAM client operating on a mobile phone. In particular, to avoid difficulties arising through the narrow field-of-view mobile phone cameras, they transmit keyframes to a server system, which then performs localization based on a pre-made and geo-registered model of the environment. The SLAM client receives updates in form of pose corrections and anchor points to refine the local map and limit drifting. The system allows globally registered tracking in real-time and is not limited to stored maps, however, the server-side reconstruction is not altered over time.

A system to combine traditional keyframe-based SLAM with panorama (rotation only) mapping and tracking, is presented by Gauglitz in [58]. Key aspect is to identify the camera motion and to switch between the two models, based on the introduced geometric

robust information criterion (GRIC). If tracking is lost, a new track is started and tried to be stitched to previous tracks by detecting overlapping regions in images, instead of trying to re-localize the system and possibly rejecting information, which is not used for re-localization. The result is a combination of a 3D structure and panoramic maps.

Sweeney describes in [56], a method to use SfM and SLAM cooperatively to enhance each others performance, without implementation. It allows SLAM systems, to navigate within pre-made 3D models or to discover new areas. Further, multiple users will be able to extend and enhance reconstructions of urban environments. Those users will provide visual data to fill coverage gaps and receive 3D information for local usage. He also mentions, that it is necessary to reduce costs of bundle adjustment for large-scale dense 3D models, however, any proof of concept is still missing to date.

Hook *et al.* show in [29], different methods for server-client and/or client-client communication for map exchange, generated with devices, which are capable for image retrieval and data communication, as well as sensors like GPS or magnetometers. Exchanged maps between clients and server may be stitched, by identifying common features in maps and calculating rotation, translation and scale between them, as well as using other sensors to ease the calculation. Clients may then receive updates, to localize themselves in the stitched maps. If such a map exceeds the local storage of a client, the server may handle the map as multiple sub-maps, and update one client with one of those. Further, they show different data transmission scenarios, like whole maps, reduced subsets of maps, necessary data, compressed maps or maps with excluded data, which can be recalculated.

3.4 Comparison to our approach

In contrast to our proposed system, some of the mentioned works point in similar directions. [29, 37, 56, 62] describe server-client architectures and how to combine maps generated by clients or multiple cameras of the same system. Riazuelos C^2TAM puts the tracking on the clients and the mapping to the server, generating a hard dependency between clients and server. Instead of trying to update whole maps on clients like Sweeney and Hook, we let clients decide by their own, what information to get and limit the given possibilities to poses and keyframes. We also highlight a loose dependency between server and clients and support different clients with no restriction, except the implementation of communication methods. PTAMM shows a method, to switch between maps without the need of stitching, by detecting edge areas of the active map.

4 Our Approach

This chapter describes our approach. We go through the server system and describe different clients and relevant scenarios challenging servers and clients. Further we show in detail, how individual parts of the server work and provide reasonable solutions for occurring problems.

4.1 SfM Pipeline

As mentioned in the previous section, the used SfM pipeline is based on Hoppe's work[12]. The keyframe based reconstruction pipeline (see figure 4.1 and 4.2) is capable to generate a full 3D reconstruction for a set of images and provided camera calibrations. To avoid side effects of uncalibrated cameras, the system operates with calibrated cameras only, otherwise an estimation over keyframe pairs was necessary. The 5-Point-Pose algorithm (see [42] and [52]) is used for initialization and applied between the first two keyframes. Further keyframes, are added using the 3-Point-Pose [43] algorithm. All 2D-2D correspondences are calculated using the GPUSIFT implementation, described in [48] verified using epipolar geometry constraints. 3D points are triangulated based on valid 2D-2D correspondences between pairs of sequential keyframes. A repetitive bundle adjustment step (see section 2.7) is applied to minimize the error in the reconstruction and to reduce drift, which is caused through the incremental nature of the reconstruction approach.

The result of the SfM pipeline is a 3D representation of the observed environment, including the camera poses. Such a map holds a sparse 3D point cloud, corresponding features to the 3D points, poses of the used keyframes and the used keyframes themselves. As described, a keyframe runs through the following steps, to extend a client specific reconstruction.

1 A Keyframe is added to the pipeline.

2 SIFT features of the added keyframe are extracted.

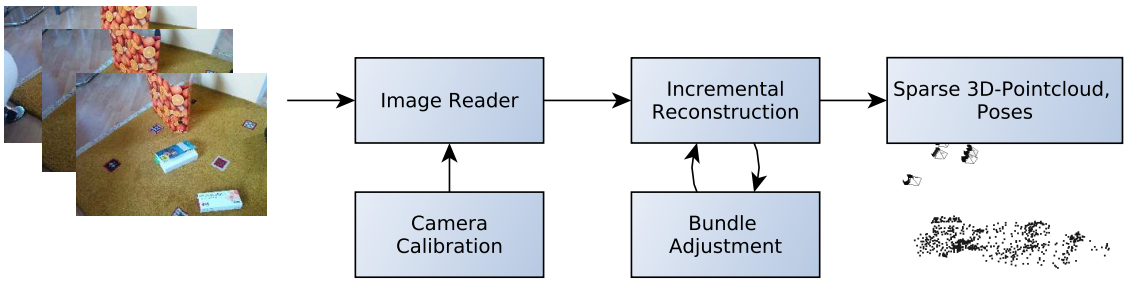
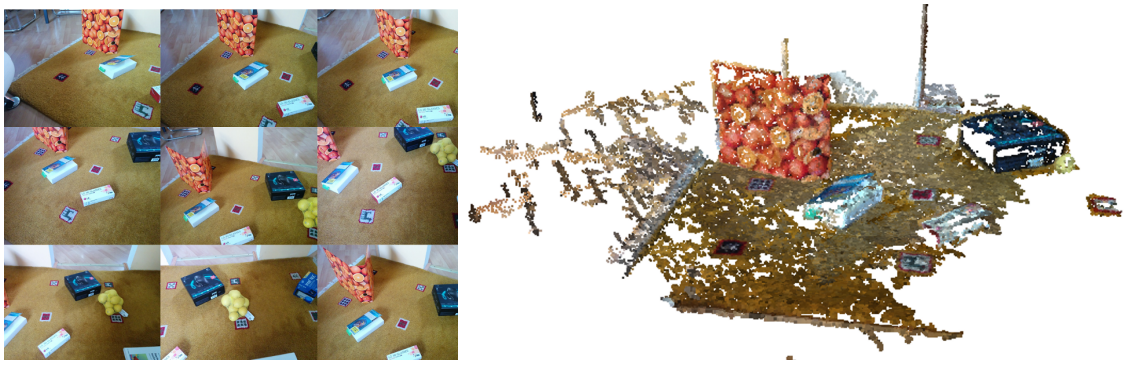


Figure 4.1: Keyframe based reconstruction pipeline. Keyframes and camera calibration are passed to the incremental reconstruction. After adding a new camera a bundle adjustment step is applied resulting in a sparse 3D point cloud including the camera poses.



(a) **(b)**

Figure 4.2: Scene *Carpet*. **(a)** shows nine out of twelve input keyframes. **(b)** shows the reconstructed scene. A densification algorithm like PVMS[24],[23] is applied to the sparse 3D point cloud.

3 The epipolar geometry is validated, based on the SIFT features.

4 If the keyframe is the second one, 2D-2D correspondences are searched between the first two keyframes and the 5PP algorithm is used for initialization, otherwise 2D-3D correspondences between the added keyframe and the stored features are identified, and the 3PP algorithm is used to add the keyframe to the map. The map is extended by new 3D points, the corresponding features and the calculated pose.

5 At this point, a new keyframe with corresponding pose is added and the map is expanded with new 3D points, based on the SIFT features of step *2*.

6 The bundle adjustment step optimizes all poses and 3D points.

7 The pipeline waits for new keyframes.

Our proposed server system, which is described in the following section, uses one SfM pipeline per client, as described above. Therefore, each client uses its own pipeline, which simplifies the problem of managing multiple reconstructions.

4.2 Server

The Server keeps track of all clients and tries to detect overlapping regions among keyframes. The aim is to combine different clients, to produce larger maps. The client registration process requires the clients to provide their camera calibration information. Further, clients can commit their initial baseline length for scaling purposes and keyframe related poses for client-space transformation purposes to support different client setups and transform between them. Allowing clients to commit their own calibrations and other information turns our server into a very versatile system and enables the usage of a wide range of various clients.

A server-side client session is initialized after receiving the camera calibration. Each session holds a SfM pipeline, including information like initial baseline distance, keyframes and related poses. After initialization, each client can commit his keyframes and poses to extend the server-side reconstruction. Figure 4.3 shows the main parts of the server system, including paths for data exchange with multiple clients. To detect overlaps among images, two clients are chosen and their keyframes are matched against each other based on natural features. Either a new map is introduced or an existing one is enlarged. Maps in the pool keep pace with the client reconstructions and are refreshed, as client reconstructions grow. Managing stored maps, includes preparing keyframes and poses for receiving clients and maintaining client related pull queues. The preparation of client data contains transformation of clients between their coordinate systems and their stored maps. In particular, stored maps are re-centered to align with a specific client space, or correction terms are added before sending data. Correction terms are needed to compensate for deviating systems e.g. when clients employ their own bundle adjustment algorithms. The following sections will describe the required parts for the server to operate properly.

4.2.1 Detection of image overlaps and merge calculation

The detection of overlaps is a key aspect to our implementation. It decides, upon the creation of new maps, and when existing ones should be extended. Validations among all possible client pairs are regularly done. In particular, different strategies are possible to select a client pair. For instance, if GPS data is available, one could generate client pairs based on the distance between them. Kreyframe based GPS data, could be used to improve the validation order of keyframe pairs, according to the distance between keyframes, to decrease the number needed attempts, until a valid keyframe pair is found. This might be relevant for city-scale reconstructions, with a large number of participating clients and many keyframes.

For simplicity, we iterate over all clients and create client pairs according to following conditions:

(i) Clients are not participating to the same map.

(ii) Clients have a valid reconstruction.

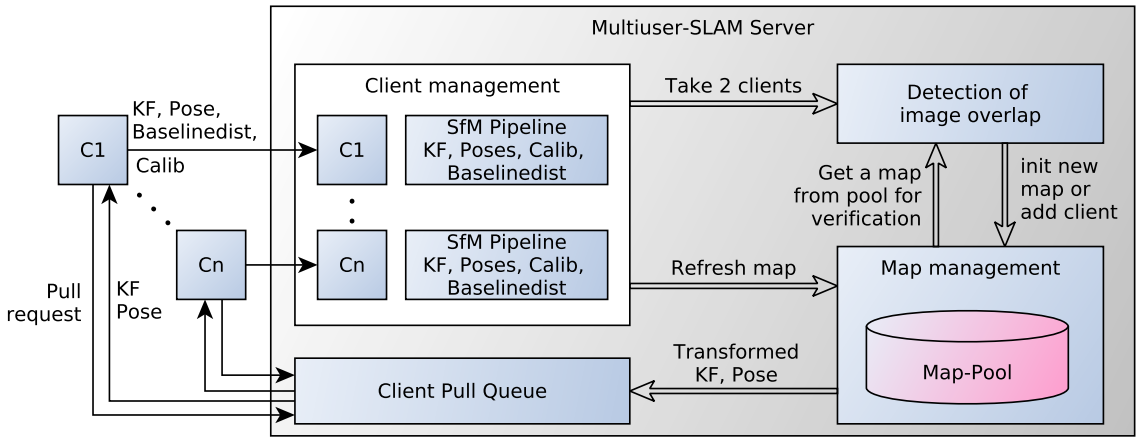


Figure 4.3: Server structure. *C1 ... Cn* represent clients. A client commits poses and keyframes, an initial baseline-distance and the necessary camera calibration (*Calib*). The generated instance (*C1* within *Client management*), is than used to detect overlapping regions in images. On success, a new map will be generated or a new client extends an existing map (*Map management*). The *Map management* will continuously check if participating clients have grown and update their map. Afterwards the *Client Pull Queue* is updated and clients can pull new poses and keyframes to update or improve their own local systems.

(iii) Clients have at least four keyframes committed, to increase the probability of having enough 3D points in the observed area, based on the selected client pair, the system selects one committed keyframe of each client for further validation.

A specific keyframe pair of two clients, is only checked once, because the image based matching will not deliver different results when matched multiple times. By marking already checked keyframes pairs and excluding them from further checks, the number of keyframe pairs to check remains small compared to the number of committed keyframes. If the detection is successful, the server calculates the merge which is needed to combine two clients, resulting in a new or a larger stored map. Afterwards, the process is repeated, until all clients are combined. Image matches are searched between the following client pairs:

(i) Two clients with no map participation.

(ii) Two clients participating to different maps.

(iii) One client with no participation and one client belonging to a certain combined map.

Otherwise the calculation for the current client pair is skipped. The ideal result is, to generate a single combined map, wherein all clients reside. A more common case, when it comes to large-scale reconstructions, is, that a group of clients create a combined map, another group creates another combined map and some clients remain without any participation.

The overlap detection is based on feature-based image matching using the GPU implementation of SIFT [39, 48]. Only already extracted features are used for the matching

operation (features already extracted by the SfM pipeline at the time, when the keyframe was added). This saves computation time, and for a next step, more important, by maintaining 2D-3D correspondences within clients. Epipolar geometry is used to verify the image based matching result. At success, we can obtain 2D-3D or 3D-3D correspondeces between two clients and use them as input to calculate the merge. Therefore, we provide two methods. Namely the *Horn Alignment* which operates on 3D-3D correspondences and the *3-Point-Pose and Scale Estimation* method. Figure 4.5 shows the image based matching result and related merged clients.

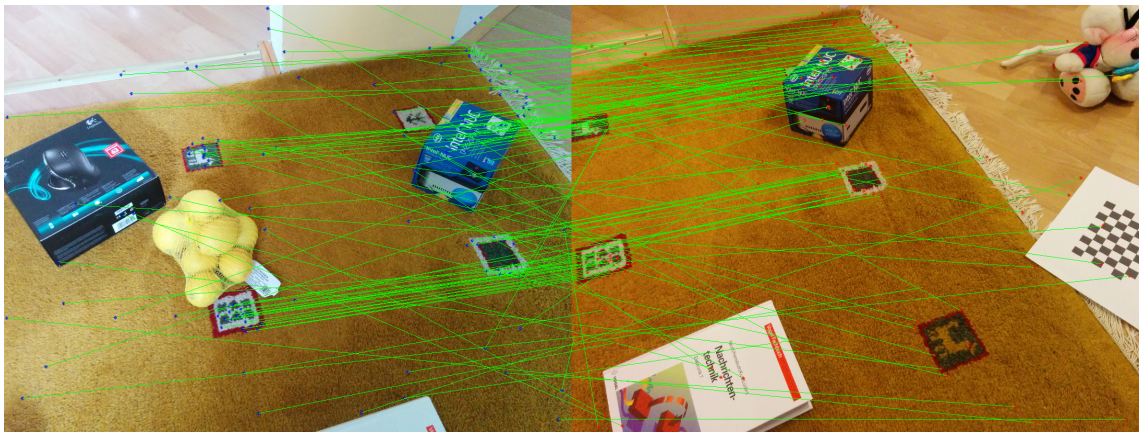
On success of either previous merge attempts, the following scenarios are possible. If none of the clients participate to a map in the map-pool, a new map is created. If one of the clients participates to a map, then the map is expanded, using the information of the other client. If both participate to the same map, no detection of overlapping keyframes is needed and the next client pair can be processed. Maps are merged, if the clients participate to different maps.

The map pool stores all combined maps in the form of a list, but it is also possible to use GPS-like information to order maps according to their real world position, if fast geo-location depended access is needed (e.g. in city-scale reconstructions). The obvious reason is, to quickly determine nearby maps for lost clients or initial updates. Whenever two clients with no map participation are combined, a new map is added to the pool. Each map in the pool holds a list of involved clients, their transformation to the client (always the first initial client) of the map, and the size of their actual map, which is used to quickly determine if a refresh of the merged map is needed. Additionally, each map can refresh itself with new client information, as described in section 4.2.2.

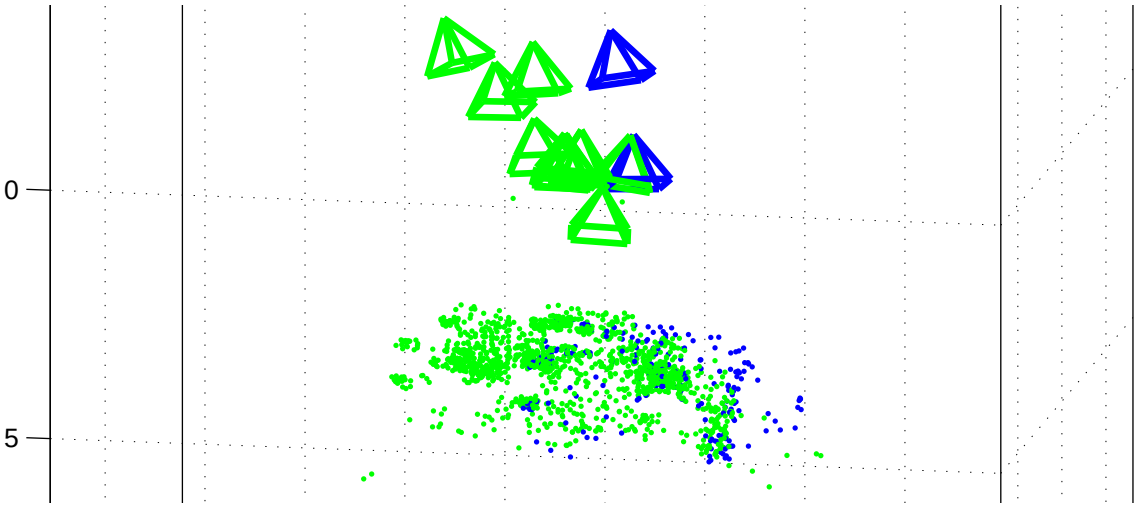
Horn alignment

We decribe the algorithm in section 2.6, and Horn shows in his work [30] a closed form solution. The input of the algorithm are two corresponding sets of 3D points, and the result of the aligment is the transformation between the two input 3D point sets. In our approach, it is necessary to wrap the Horn algorithm into a RANSAC [22] loop, because of possible outliers, which would corrupt the result. Using the Horn result, we can easily transform one client (poses and 3D points) into another clients space (see equations 2.16 and 2.17).

In our specific case, an overlap of at least one frame is needed, to provide enough 3D points in both clients. The reason is that enough feature points of the overlapping area have to be seen from both clients ideally in multiple frames for a correct result. With a large number of potential point-point correspondences and a high number of entries, the calculation of a merge can take a considerable amount of time. This influences how fast related clients receive new updates. In edge cases, it can be



(a) Image matching



(b) Merge result

Figure 4.4: Scene *Carpet*. **(a)** shows the detected image overlap using SIFT features and **(b)** shows the calculated merge between to clients. The green points and frustums stem from the initial client and all cameras of the blue client have been correctly transformed into the green clients space.

the difference between tracking and tracking loss, which might disturb a user of an AR application. An advantage of this method is, that it takes care of the scale by default, and transformations in both directions are very simple to apply. On the downside, the quality of the result of the algorithm strongly depends on both estimated sets of 3D points, which again are based on the accuracy of the detected natural features.

3-Point-Pose and Scale Estimation

In this section we use an extended notation, where the superscript shows the 3D space e.g. X^A refers to a 3D point in the 3D space of a certain client A.

3PP uses 3D-2D correspondences to calculate the transformation between a

set of 3D point and corresponding 2D observations in an image (see section 2.5). Therefore, the inputs are 3D points of one client (A) with the matching 2D points (feature points) of the other client (B). In practice, we take detected matches between two images, which provide 2D-2D point correspondences and search the according 3D points of one client. The delivered result of the 3PP algorithm is a new Pose ($\mathbf{P}^A_{3PP}$) of the involved camera ($\mathbf{P}^B_m$) of client B in the 3D space of client A, as shown in equation 4.1, where $\mathbf{X}^A$ represents the stacked 3D points of client A and $\mathbf{x}^{\mathbf{P}^B_m}$ the stacked 2D points of the involved camera of client B. 3D points are transformed using equation 4.3, with the idea to first move the point into the coordinate system of the involved camera (equation 4.2), and afterwards to transform the point into the 3D space of client A, using the 3PP result. The aim is to represent 3D points of client B (X^B) in the space of client A (X^A). s determines the scale, which is explained later in this section.

$$\mathbf{P}^A_{3PP} = 3PP(\mathbf{X}^A, \mathbf{x}^{\mathbf{P}^B_m}) \tag{4.1}$$

$$\mathrm{X}^{\mathbf{P}^B_m} = \mathbf{P}^B_m * \begin{pmatrix} \mathrm{X}^B \\ 1 \end{pmatrix} \tag{4.2}$$

$$\mathrm{X}^A = (\mathbf{P}^A_{3PP})^{-1} * \begin{pmatrix} \mathrm{X}^{\mathbf{P}^B_m} * s \\ 1 \end{pmatrix} \tag{4.3}$$

Poses can be transformed using equation 4.4, where $\mathbf{P}^B_1, \ldots, \mathbf{P}^B_n$ are poses of client B, which are transformed to the equivalent poses in 3D space of client A ($\mathbf{P}^A_{B1}, \ldots, \mathbf{P}^A_{Bn}$). The scale parameter only contributes to the 3PP merge result. The idea is similar to the point tranformation, where we first transform the poses into the coordinate system of the involved camera($\mathbf{P}^B_m$) and afterwards into the 3D space of the other client.

$$\begin{aligned} \mathbf{P}^A_{B1} &= \mathbf{P}^B_1 * \begin{bmatrix} (\mathbf{P}^B_m)^{-1} & \\ \mathbf{0}^T & 1 \end{bmatrix} * \begin{bmatrix} \mathbf{P}^A_{3PP} & \\ \mathbf{0}^T & s \end{bmatrix} \\ &\vdots \\ \mathbf{P}^A_{Bn} &= \mathbf{P}^B_n * \begin{bmatrix} (\mathbf{P}^B_m)^{-1} & \\ \mathbf{0}^T & 1 \end{bmatrix} * \begin{bmatrix} \mathbf{P}^A_{3PP} & \\ \mathbf{0}^T & s \end{bmatrix} \end{aligned} \tag{4.4}$$

As Nistér mentions in his paper [43], far more than three 3D-2D point correspondences are needed for robust estimates and to avoid dealing with multiple solutions. However, the algorithm provides rotation and translation, therefore we have to additionally calculate the scale s, based on a single 3D-3D correspondence. The scale can be calculated as the ratio of the distance between a 3D point in space A and the merge result (involved camera in space of client A) and the distance of the corresponding 3D point in space B and the involved camera. Equation 4.5 shows, how to determine s where $\mathbf{c}$ is the camera center and $||\cdot||$ is the euclidean distance, using a single point correspondence. A single 3D-3D

match would be inaccurate, therefore we can take multiple matches and calculate the median over all determined scales. Only few matches are sufficient for good estimates, in our implementation.

$$s = \frac{\left|\left|\mathrm{X}^A - \mathbf{c}^A_{3PP}\right|\right|}{||\mathrm{X}^B - \mathbf{c}^B_m||} \tag{4.5}$$

In our cases 3PP, outperforms Horn, because it uses 3D-2D rather than 3D-3D correspondences, minimizing the chances of bad estimated 3D points in one set. Further, one client does not need a full reconstruction of the overlapping section and therefore, 3PP is less restrictive. Imagine that client A and B are moving towards each other. The Horn alignment, will have to wait until enough points of the overlapping region have been reconstructed to successfully calculate the merge (fewer points at edge-regions because of missing follow-up frames e.g. at sharp turns). This could last a couple of frames until the reconstruction of the affected area grows. The 3PP approach is able to calculate the merge, if one of both clients has some 3D points of this region. Eventually the initial scale estimate is a bit off, but it can be re-estimated, with little effort. Further, it is more stable in terms of outliers because it only remains on one set of estimated 3D points and not of two sets, as the Horn algorithm.

A valid merge calculation generates a so called merged-map, which initially will include the two clients involved. If another client generates a valid merge calculation to a participant of a merged-map, the merged-map is extended. A merged-map always resides in the space of the first client. If new clients are added, they can either be directly connected to the first participant of the map or to another participants, which then may be directly or indirectly connected to the first one. The first case is simple: just apply the merge result as shown above. The second case is more complex, because a chain of connected clients arises.

For instance, let's assume that we have three clients A,B and C. B is connected to A and C to B. The merged-map holds A and B, with A as origin. To add C, we have to transform it first to B, using the merge result of B-C and subsequently, we have to apply the merge result of A-B to C, to transform C into A's space. Figure 4.5 visualizes a typical scenario of three clients.

4.2.2 Refreshing stored maps

Refreshing stored maps is important, because it serves as a basis for the update procedure of clients belonging to a certain combined map. Without refresh, client specific pull-queues can not be filled. When clients explore new areas of their environment, they simultaneously generate new updates for clients participating to the same map.

During the implementation, we faced two applicable methods. One method

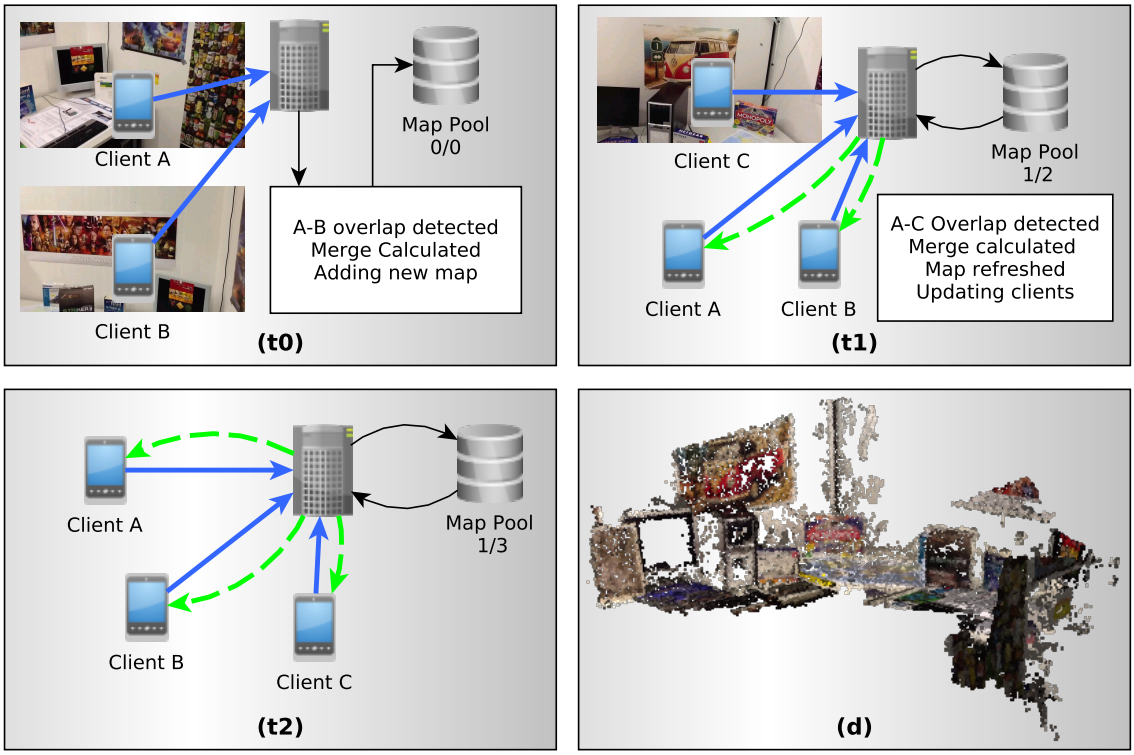


Figure 4.5: Scene *Office*. Server map generation. *Blue* arrows visualize data sent from client to server and *green* ones the opposite direction. The number beneath the map pool determines the number of the current map *(1/*)*and the amount of participating clients *(*/2)*. **(t0)** shows clients A and B connecting to the server and transmitting their data, including multiple keyframes. After the detected image overlap, a new map is added to the map pool. At **(t1)** the server system maintains the map pool, by refreshing it with data from A, B and answering pull requests from combined clients. Client C is added to the scene and the existing map between A and B is expanded. **(t2)** shows, that now pull requests of all clients are answered and the existing map is refreshed, by all participating clients. **(d)** shows the final reconstruction from combined information of all three clients.

is, to discard the actual map and to re-create it including the new committed key frames, while using the already calculated merge information. The other method is, to execute delta updates of occurred changes, which includes, the detection of changes in affected maps. The advantage of the first method is that bundle adjustment of the whole map can be avoided. In detail, at every refresh, scales between involved clients are re-estimated, to compensate for the client's bundle adjustment. Afterwards the map is built from scratch by merging whole clients into the map. Equivalent 3D points are fused to one 3D point. This approach is not as complex as updating the changes, and is fast enough when operated as an own thread. Additionally, the refresh procedure could be triggered when one or more pull-queues are almost empty, to avoid empty queues and to possibly reduce the the number of refreshes done.

The second method applies bundle adjustment on the whole map and only adopts new points, using a distance measurement (if an existing point and a new one are close

enough, they are handled as the same point). Bundle adjustment tends to get slower with bigger maps and in our case, above a certain map size it would delay the refresh process, which may be negligible with one or two stored maps, but it would get worse with more maps. Delaying the refresh process, also delays when pull-queues are refilled and this translates one to one in later client updates in the case of empty queues before update.

In both cases, maps are only refreshed, if clients grow and the merge result is used to add new information, where the first attempt always has to work to re-initialize the combined map, and the second attempt has only to adopt new parts, but also to apply bundle adjustment on the combined map.

4.2.3 Client pose preparation and offset correction

With the ability to use different clients, we have to handle diverging SLAM-systems. Therefore we can name several difficulties. (i) Scale, origin and orientation mismatch between a certain client and server map. (ii) Different bundle adjustment algorithms, have different influences as maps grow over time. One reason is that the server may start using a different initialization as the client (e.g. first added server camera looks into positive z-axis and the related client camera looks into negative z-axis).

The scale problem can be solved, if the the client commits its initial baseline distance, but it is only a better estimation and could be off after some time of operation according to the local bundle adjustment. The server can scale the corresponding pose-pair of the merged-map to the clients baseline distance. If the client commits the first pose, then the server is able to transform the merged-map, using equation 4.7. Re-centering can be done with the same equation. Each client resides in its own coordinate system. The resulting merged-map shares the coordinate system with the first participant. All other clients reside relative to the first one. To enable pose-processing on each client, the map has to be re-centered for each client, based on his first pose.

$$\mathbf{P}_{diff} = \begin{bmatrix} \mathbf{P}^{-1}_{neworigin} & \\ \mathbf{0}^T & 1 \end{bmatrix} * \begin{bmatrix} \mathbf{P}_{actualorigin} & \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4.6}$$

$$\mathbf{P}^{new}_n = \mathbf{P}^{actual}_n * \mathbf{P}_{diff} \tag{4.7}$$

The offset correction is needed, because of different SLAM-systems and different bundle adjustment algorithms used. It happens, that above a certain number of cameras, the systems show different behaviours, in terms of where to estimate the next camera. Also, poses after bundle adjustment improvements may deviate from the server ones. Clients can commit poses related to committed keyframes, to inform the server about the local circumstances. Having the local information, we can create correspondences between merged-map poses and specific client ones. In particular, after solving the previous problems (origin, scale and orientation), we get a new valid pose in client space. Knowing the

possible difference, we can find the closest known client pose, which is related to the original client pose, namely the offset-pose. The closest one is required, because of a possible growing offset over time. Using equation 4.9, we can determine the correct pose and push it into the client-pull-queue.

$$\mathbf{P}_{diff} = \begin{bmatrix} \mathbf{P}^{-1}_{closest} & \\ \mathbf{0}^T & 1 \end{bmatrix} * \begin{bmatrix} \mathbf{P}_{closestlocal} & \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4.8}$$

$$\mathbf{P}^{newcorrected}_{n} = \mathbf{P}^{new}_{n} * \mathbf{P}_{diff} \tag{4.9}$$

Figure 4.6 shows server poses, which are pulled by clients. The client correction has to be done per client and is only applied to poses. The computational effort of the applied client-pose-correction is very low, because of the matrix multiplication, but we have to hold in mind, when working with maps with thousands of poses, we will have to improve the search for the closest client pose to apply the offset pose. It could potentially form a bottleneck, when it comes to fill the pull queues.

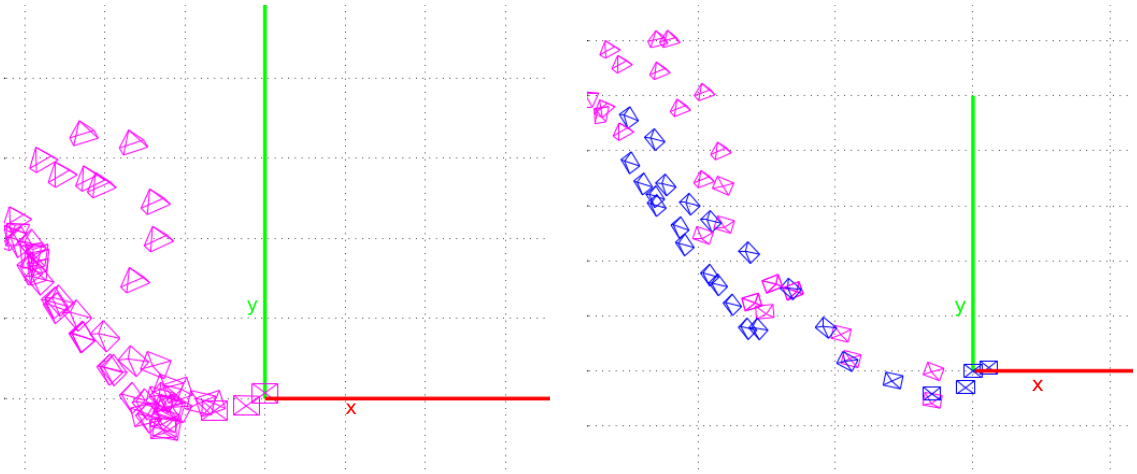
When operating with autonomous clients, which use different cameras, keyframes have to be prepared before sending them back to the client. Different cameras are easily detectable by the provided camera calibration. There are two possible scenarios. (i) The server sends undistorted keyframes and calibrations and each client handles the image itself regarding to the used implementation. (ii) The server warps keyframes specific for each client with their provided camera calibration. Therefore, the client can handle received keyframes like local ones. In thought of taking load from clients and allowing them additional computational time for more important tasks, (ii) is more applicable and existing client systems are easier to adopt.

4.3 Client

Almost every client can be used with our server implementation, ranging from very simple clients up to systems like PTAM or even more complex ones. Clients have only to implement the push and pull methods, which are used for communication, and have to transmit the required data for participation. In our case, we use a very simple client, which is committing keyframes without any SLAM-system, and a high sophisticated client, which performs local SLAM. Further, we add three different operating modes for server interaction, differing in how received data is handled and how much external information is used to improve the local map, including 3D points with related measurements and poses.

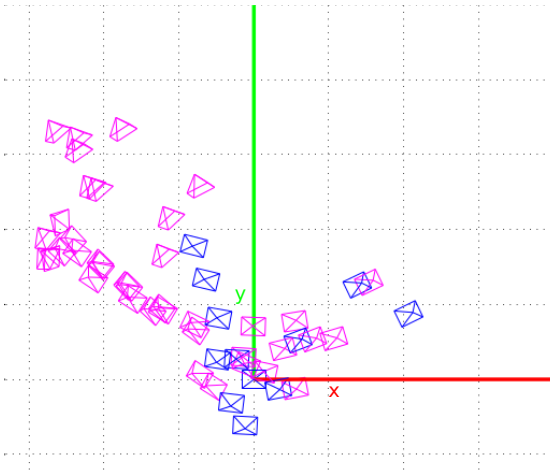
4.3.1 Thin Client, no SLAM

A thin client may perform no local SLAM or any kind of 3D related computation. Further, it is only used as an image acquisition device. The committed keyframes are used

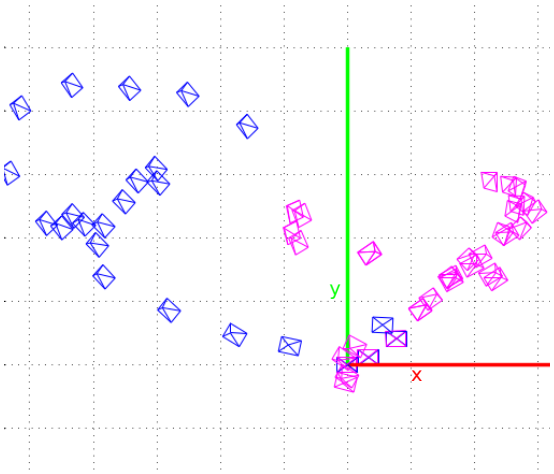


(a) Server

(b) Client A



(c) Client B



(d) Client C

Figure 4.6: Client offset correction. Scene *Cardgame*. Local poses in *blue*, received poses in *magenta*. **(a)** shows the merged map, based on three clients as created by the server. **(b)** shows the local map of client A, including received poses. Comparing **(a)** and **(b)**, we can see a difference in scale between server and client. Therefore we need to correct the poses before sending. **(c)** and **(d)** visualize the same information as **(b)** but from other clients perspective. Received poses are provided by each other client e.g. A received from B and C.

for reconstruction purposes on the server. A common use-case is to create a map or reconstruction, with higher resolution images. A possible client could be any image or video capturing handheld device, like a Google Nexus 5, with a camera operating at a resolution of 3264 $\times$ 2448 pixels. Figure 4.2 and 5.3 show example reconstructions based on images acquired with such a handheld device.

4.3.2 Fat Client

The used autonomous client performs local SLAM and can operate in real-time. The SLAM system, is based on optical flow tracking (see [8, 10, 17, 55]), which allows to estimate motion in sequential images. Basically, the optical flow tells, which image-points to select. Based on this information, custom feature points are extracted and used for triangulation. The local map stores features, 3D points, keyframes, and poses. Relating to the server, the autonomous client can commit keyframes, related poses, the initial baseline distance between the first two cameras and the used camera calibration. Further, the client can decide what to pull: keyframes or keyframes with related poses. Therefore we implemented three methods.

V1 pulls only keyframes and tries to improve the map, by adding more observations to already known feature-points.

V2 pulls keyframes with related poses and adds them to the system, if no contradiction within the existing system occurs.

V3 uses the same information as *V2*, with the addition, that new points are triangulated and added to the local map

Client-side bundle adjustment performs first on smaller parts of the map, before fulfilling the global optimization as described in section 2.7. By pre optimizing clusters of the actual map, the main bundle adjustment step over the whole map tends to finish very fast.

V1 to V3 differ in how to process received data, like keyframes or poses. A new keyframe is added when a certain amount of time has passed or when the client has significantly moved. Therefore, natural features are extracted and added to the map for tracking. Features, which are not triangulated yet but tracked (a feature, which has been seen in a previous frame and added to the tracker) are now triangulated and the resulting 3D points are added to the map. New added features for tracking, may be triangulated with the next keyframe. Based on the keyframe the actual pose is calculated. Additionally, the new keyframe and the calculated pose are committed to the server. In detail, they are added to a send-queue and processed, when the client has enough computational resources. The transmitted keyframe, will extend the client specific reconstruction and is used to detect overlapping regions of keyframes among other clients on the server. The transmitted pose is added to a keyframe related look-up table, and is used for offset and orientation correction on the server. The following sections will describe the three operational modes for clients in detail.

4.3.2.1 Client Mode: V1 - Keyframes

V1 describes the simplest of the three methods. Therefore, the client has to commit his keyframes and to pull new ones. No pose or other server data is required. A received

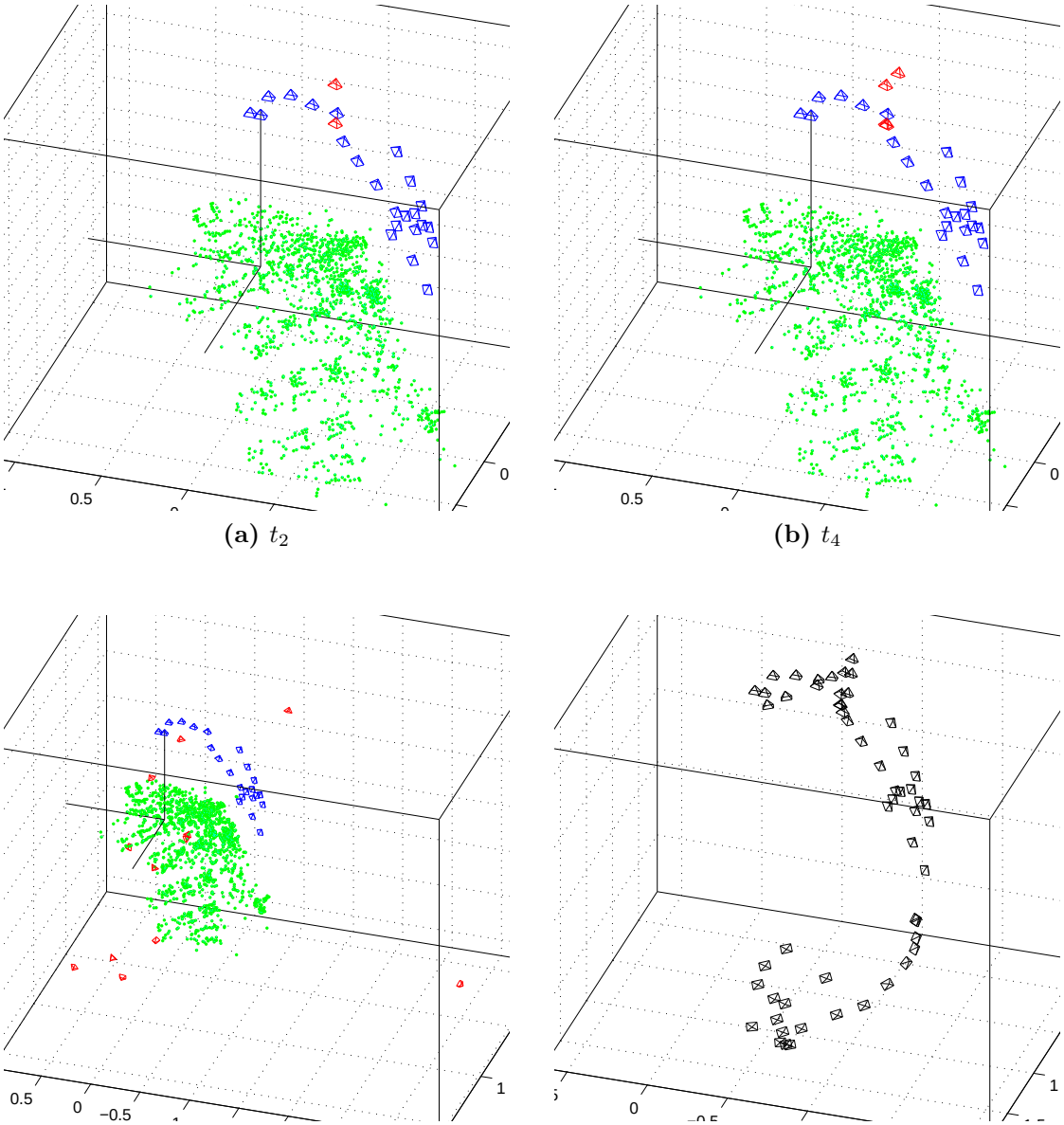
keyframe is localized within the map, using the 3PP algorithm. In particular, natural features are extracted and matched against features of 3D points of the local map, resulting in 2D-3D correspondences, which serve as input for the 3PP algorithm. This version can only improve the local map, by adding more observations to known points, rather then adding new points. New keyframes can only be added to already seen areas of the scene, because new keyframes are matched against features of existing 3D points. The 3PP algorithm returns a valid pose for the received keyframe in the local coordinate system. Figure 4.7 shows the improved local map and a comparison to a map including false positives. Because of the image based calculation we can choose very strict thresholds (number of RANSAC iterations, amount of inliers, RANSAC sample size) for the 3PP pose estimation. As a consequence, fewer keyframes are used to improve the map, but also the false positives are reduced. Therefore, the added keyframes are real improvements in terms of measurements per observed point. By lowering the thresholds, it is easily possible to corrupt the map by adding false observations to known points (see Figure 4.7, *error case*). The computational effort can be high enough to affect real-time performance and strongly depends on the amount of needed iterations of the RANSAC pose estimation and number of used 2D-3D correspondences.

4.3.2.2 Client Mode: V2 - Keyframes and related Poses

V2 tries to add incoming keyframes to the existing local map. No RANSAC pose estimation is needed, because the received pose does already fit into the clients coordinate system. The server pose is verified by projecting 3D points of the overlapping area into the received keyframe using the server pose, and measuring the re-projection error (euclidean distance between extracted feature and projected 3D point). If the error is small enough for a certain number of points, the received pose is valid. The projected 3D points are 2D-3D correspondences between features of the received keyframe and 3D points of the map, therefore natural features of the received keyframe are extracted and used to find corresponding 3D points.

In contrast to V1, 2D-3D correspondences are not used as input for the 3PP algorithm. Valid keyframes and poses are added to the local map. This method allows the client to improve the local map, without big computational effort. Further, the client can use the information provided by other clients (e.g. poses) in application-specific context and is able to spend more computational time for e.g. content visualization.

Figure 4.8 shows the integration of server-poses over time. The main advantage over V1 is, that the map can be improved with very low computational effort, without affecting the real-time performance at all. In the same time, more received keyframes can be added, compared to V1. Therefore, the client relies on the servers accuracy of the provided data. V2 works only, if the client commits the poses related to the committed keyframes. Differences between client and server poses are caused by independent bundle adjustment steps, different implementations, and by different feature point detectors. Using commit-

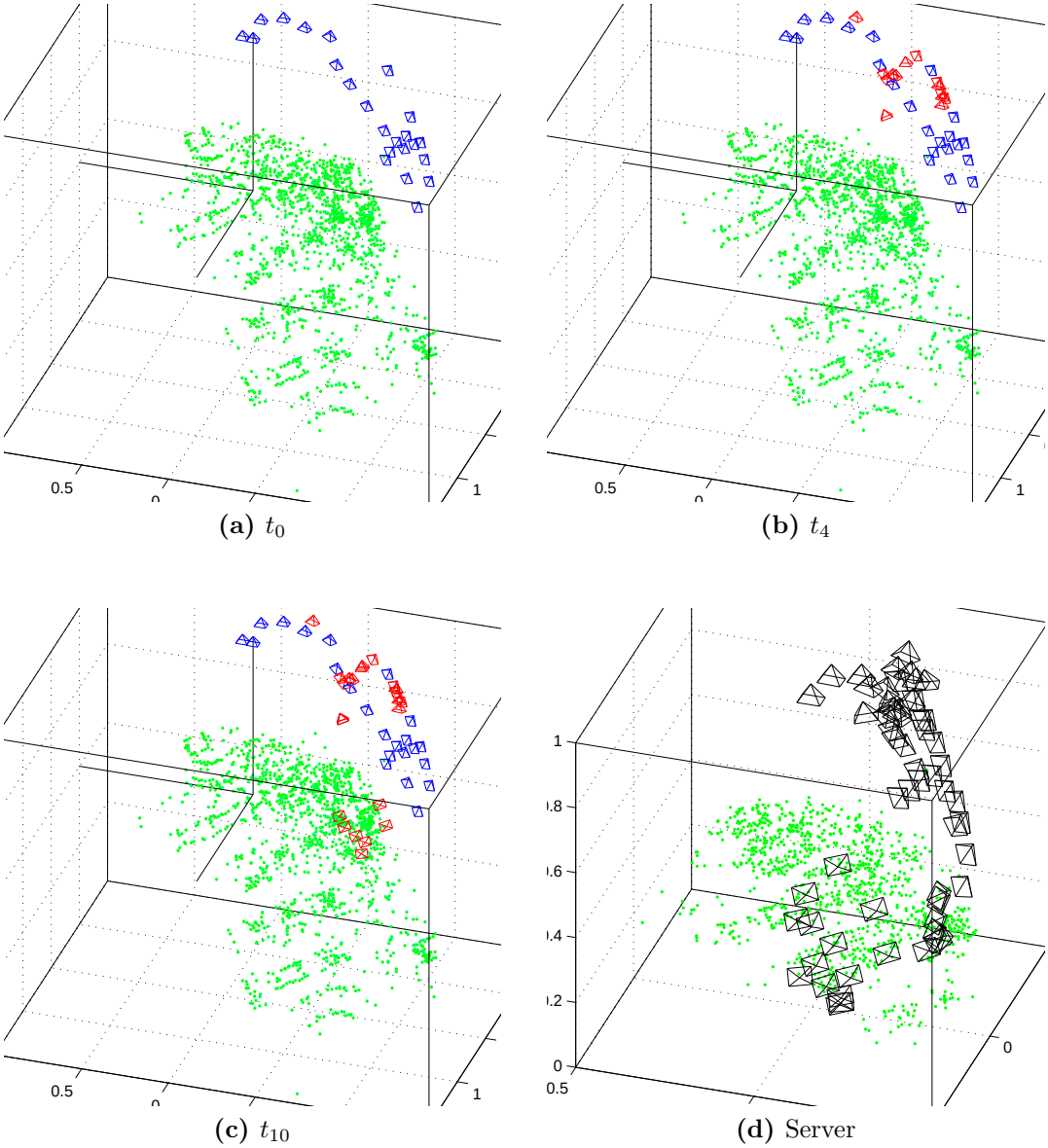


(a) t_2 **(b)** t_4

(c) error case **(d)** Server

Figure 4.7: Fat client V1. Scene *Cardgame*. Local poses in *blue*, received keyframes in *red*. **(a)** and **(b)** show the extended local map in different points in time, with reasonable position estimates. **(c)** shows the same map with low thresholds and false positives. False positives distract the system because of wrong added point observations, and decrease the tracking performance. In this case, fewer correct poses (b) are a better improvement over more partially false poses(c). **(d)** shows the representative server poses.

ted poses, the server can compensate for orientation mismatches and deviating bundle adjustment systems, to provide poses, which fit well into the local SLAM system without any contradictions to the existing map.



(a) t_0 **(b)** t_4 **(c)** t_{10} **(d)** Server

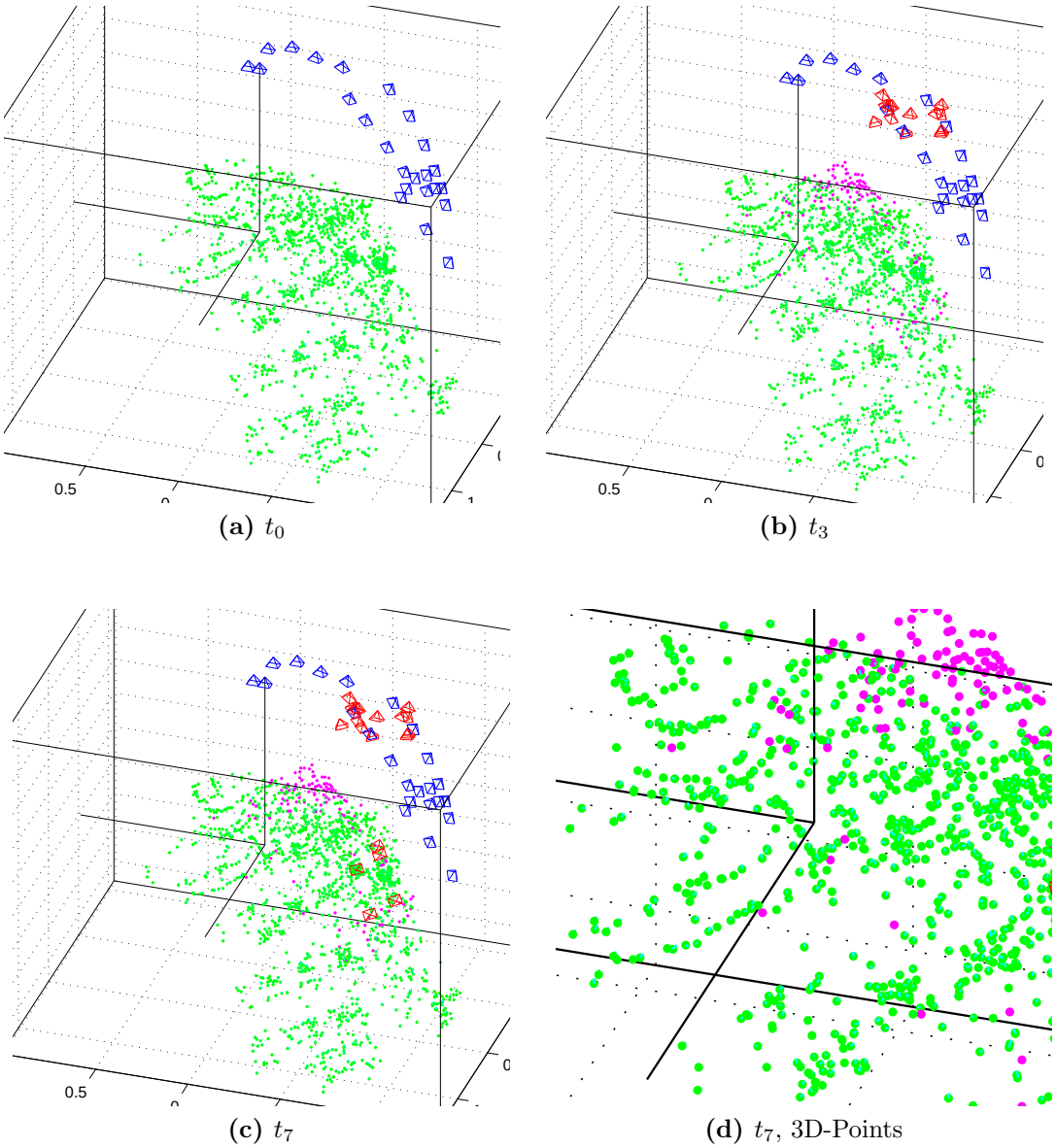
Figure 4.8: Fat Client V2. Scene *Cardgame*. Local poses in *blue*, received poses in *red*. **(a)** shows local trajectory without server information. **(b)** and **(c)** show the extended local map in different points in time. Comparing **(c)** and **(d)** shows, that the received poses (in red) fit well into the local client system and are similar to the server reconstruction with respect to different bundle adjustment steps.

4.3.2.3 Client Mode: V3 - Keyframes and related Poses, Map extension

V3 operates very similiar to V2. Keyframes and poses are used to extend the local SLAM system, where the client has to rely on estimates, provided by the server. Additionally to committed keyframes, local poses have to be committed, to improve server-pose-estimates for received poses (see *Offset correction and client data preparation* in section 4.2.3). The advantage over V2 is the additional point-triangulation, which allows map expansion.

Server keyframes are matched against local keyframes and new 3D-points are triangulated, based on these matches. In detail, natural features of the received keyframe are extracted and used to verify the received pose, as described in section 4.3.2.2 (*Client Mode: V2*). 2D-3D correspondences are used for pose verification, and 2D-2D correspondences between the received keyframe and the stored features of the map are used for point triangulation. One received keyframe is matched against all local keyframes. If there are existing 3D points for matched features based on local keyframes, the new triangulated points may not be added to the map. This avoids duplicated points and keeps the map sparse.

Figure 4.9 shows the local map behaviour over time. After some received keyframes, new points can be triangulated and added to fill unknown parts of the map. The additional computational effort of point triangulation is negligible, but it will sum up with an increasing amount of local keyframes because of the need to check received keyframes against all stored ones. This process could be speed up, when considering received poses in terms of relative position to local keyframes, to limit the search space. In our experiments, the caused delay remained within the real-time constraints. Different strategies can be used to triangulate new points, like how often a received keyframe is used for triangulation or how to reuse keyframes for additional triangulations as the client moves to expand the map. In our case, a received keyframe with pose is processed once and is then added as local keyframe, if valid. Depending on when the triangulation happens, the amount of added points may vary, because the triangulation is only done once. A more complex approach is to mark triangulated points in a certain received keyframe, and to start the triangulation a second time after the local map has changed e.g. more key frames are added or more points are triangulated. Of course the memory footprint would increase compared to the simpler way, but more 3D points would be added, and received keyframes would be used more efficient (how many points can we extract from unseen areas).



(a) t_0 **(b)** t_3 **(c)** t_7 **(d)** t_7, 3D-Points

Figure 4.9: Fat Client V3. Scene *Cardgame*. Local poses in *blue*, received poses in *red*. **(a)** shows the local trajectory, without server information. **(b)** and **(c)** show the extended local map in different points in time. *Note*: At t_7 not all server poses and keyfrarames are received yet. **(d)** is an enlarged region of (c) and shows additional 3D-Points in *magenta* based on server information at t_7. The amount of new triangulated points in this scene at t_7 is about 10 percent of the whole scene. The new triangulated points (magenta) fill an unknown part of the local map (green points, top right corner of the reconstruction).

4.4 Implementation details

4.4.1 Server

The server system is devided into two parts. One part operates as described in section 4.2 and the second part manages the communication using ZeroC-Ice[1], which is a CORBA style remote procedure call system. Further it is very versatile and supports implementations for different languages, like C/C++/Java and mobile platforms, like Android or iOS. This concept increases the flexibility of our *Multiuser-SLAM* server. It could easily be integrated into the main application, but it would decrease the exchange ability. Further, the communication part is easily exchangeable with any other suitable technology.

Poses provided by the server and client, are in the form of a 6×1 vectors, as shown in equation 4.10, representing a SE3 group of the Lie-Algebra (see section 2.8). Using the SE3 group, reduces the amount of data needed to be sent by 50%. It is just a small improvement, but it is mentionable concerning large maps including many clients.

$$\mathbf{P} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{pmatrix} \tag{4.10}$$

A merged map holds a list of participating clients, with information about relationships among clients, in particular which client is connected to the initial one. Therefore, the amount of transformations stored is reduced to one per client reducing the memory footprint, but slightly increasing the computational effort. A client which is not directly connected to the initial client, is part of a chain of transformations. Such a chain starts with a directly connected client and is enlarged, when indirectly connected clients are added to the map (detected overlap between a directly connected client and a new one). For those clients the whole chain of transformations is applied to transform them into the space of the initial client. This allows to fix individual scales at refresh time easily, without the need of recalculating a whole transformation, for one not directly connected client.

Two maps are merged, if an image overlap between clients of different maps is detected. With a valid merge result, all clients are moved into one map by simply applying the calculated merge result to the origin of the other map. Because of the previously mentioned chain of transformations among participating clients, the process of merging two maps remains simple.

[1] `https://zeroc.com/`

4.4.2 Client

The fat client can decide in which quality to provide keyframes. Each keyframes can be compressed, using the libjpeg[2]. Further, color-images can be reduced to grayscale images, to reduce the data amount approximately by two thirds, which is helpful given low bandwidth server uplinks.

Receiving keyframes is exclusively handled by clients. Each client runs an asynchronous pull loop to get new data from the server. The server does not start communication actively. Therefore, clients can handle data connections on their own, without the need to reply to server calls. Further, it allows to load-balance clients and to only pull data if they have enough resources.

4.4.3 Communication and Events

Loose dependencies between client and server allow for flexible communication. Clients can decide what to commit and what to pull. The one criteria of participation is to commit the camera calibration, but this alone will not allow the full exploration of server functions. Additionally, clients may commit their initial baseline and poses related to keyframes. The following list will go through possible server events, which may be caused by clients or internal operations. Figure 4.10 shows example communication, with one server and two clients, including the occurred server events while interacting with clients.

- **Calibration committed**: A new client is created with an own reconstruction instance and an image reading thread.
- **Initial baseline distance committed**: The initial baseline distance is added and used to scale poses, before adding them to the pull-queue.
- **Keyframe committed**: The keyframe is read by the image reading thread and added to the reconstruction instance.
- **Pose committed**: If the related frame is used in the reconstruction, the pose is added as local correspondence. If not, it remains in the list and will be checked again.
- **Overlap detected and merge calculation**: After detection, the overlap is validated using epipolar geometry and a merge is calculated. If successful, a new client is added to an existing map or a new map is initialized using the involved clients.
- **Map refresh needed**: Maps can regularly check for client changes. If so, new information is merged into the existing map.

[2] `http://libjpeg.sourceforge.net/`

- **Pull Request**: When receiving a pull-request, the server will serve the next keyframe or the next pose and add them to a client specific history. The queue could be sorted in different ways, for instance by distance from the polling client.

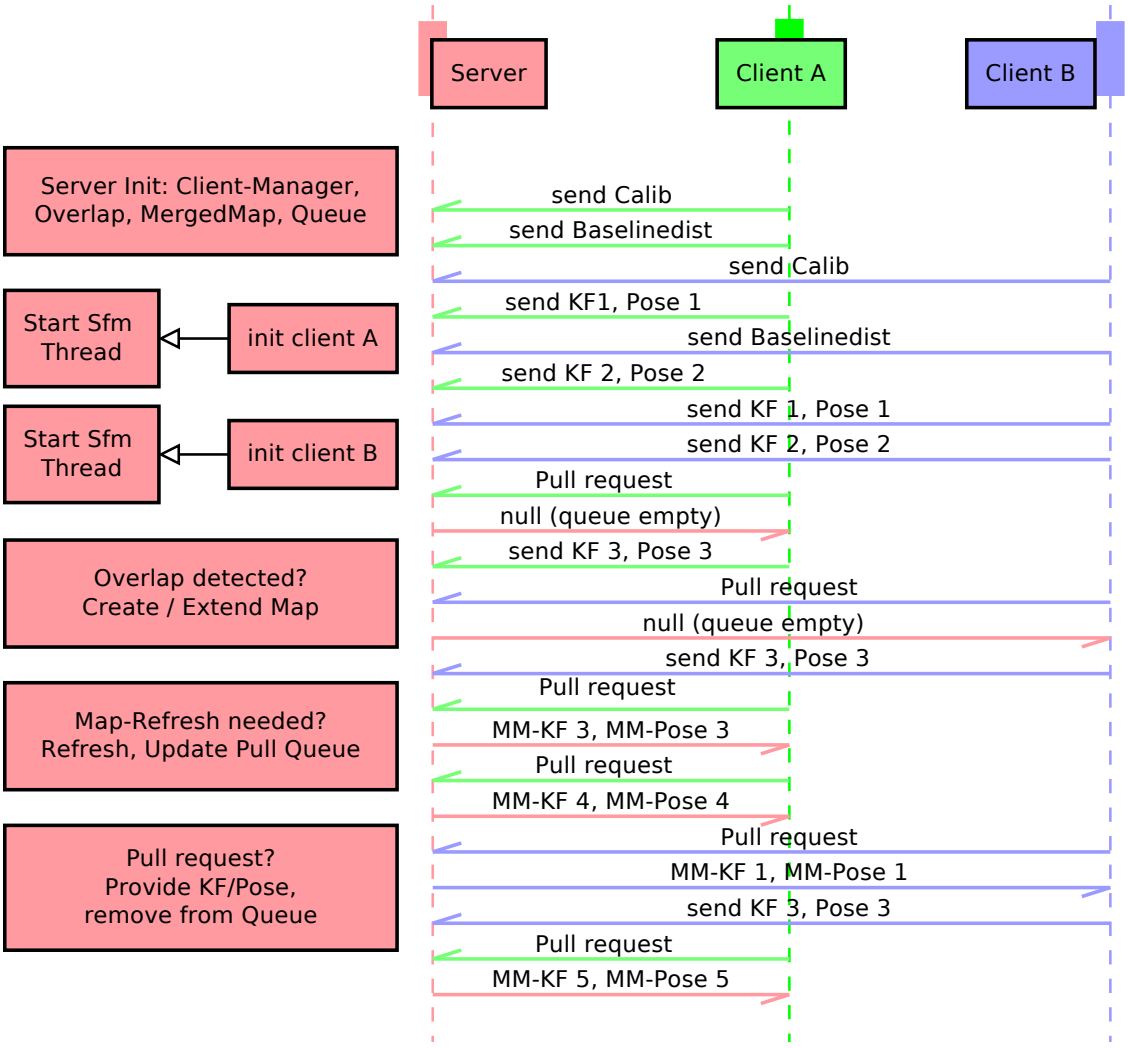


Figure 4.10: Client server communication. Example clients A and B are starting individual SLAM sessions. The server creates individual sessions for A and B. First pull-requests result in null, because there was no overlap detected. If a merged-map is created, the per client pull queues are filled and the pull-request will return a keyframe and/or a pose. *Note*: Clients can decide what to pull. The left column shows server-events and reactions, which are triggered either by clients or by internal threads (e.g. merged-map refresh thread)

5 Results

This chapter describes the scenarios and the experimental results obtained through our system. Further, we compare thin and fat clients operating in different modes, namely *V1*, *V2* and *V3*. Table 5.1 shows our test-setup including two Windows 8 Tablets operating as fat clients and Table 5.2 defines common abbreviations for this chapter. Section 5.1 shows reconstructions of four thin clients including the merge result. Section 5.2 presents fat clients operating in different modes and section 5.3 shows a general experiment in an office like environment.

Device	Specification	Op. Mode
Desktop PC	Manjaro-Linux, Intel i5 4×2.67GHz, 8GB Ram, Nvidia GTX 650	Server
Google Nexus 5	Snapdragon 800, 2GB Ram	Thin Client
Samsung XE700T	Win8.1, Intel i5 3317U 2×1.70-2.6GHz, 4GB Ram, Intel HD 3000	Auton. Client
Surface Pro 3	Win8.1, Intel i7 4650U 4×1.70-3.3GHz, 8GB Ram, Intel HD 5000	Auton. Client

Table 5.1: Devices used to test our *Multiuser-SLAM* system.

Abbrev.	Explanation
t_n	Particular point in time
#KF	Number of Keyframes
#PT	Number of 3D Points
#OB	Number of Observations

Table 5.2: Abbreviations used across the result section.

5.1 Scene Carpet

The *Carpet* scene is mainly used to verify the implemented merging approach, and to show the server capability of reconstructing a scene with higher resolution images as typically used in SLAM systems. As thin client serves a Google Nexus 5 and the committed images have a resolution of 3264 × 2448 pixels. Figure 5.3 and 5.2 show densified 3D point clouds

and the input images used. Table 5.3 describes clients and their amount of participation to the merged map. Similarly, figure 5.1 highlights how clients participate to the merged map. Reconstructions with higher resolution images have longer computation times and a very high memory consumption, compared to reconstructions based on images acquired using SLAM systems, however, they deliver a 3D model with higher level of detail. In figure 5.2, no relevant scale or transformation issues are observable.

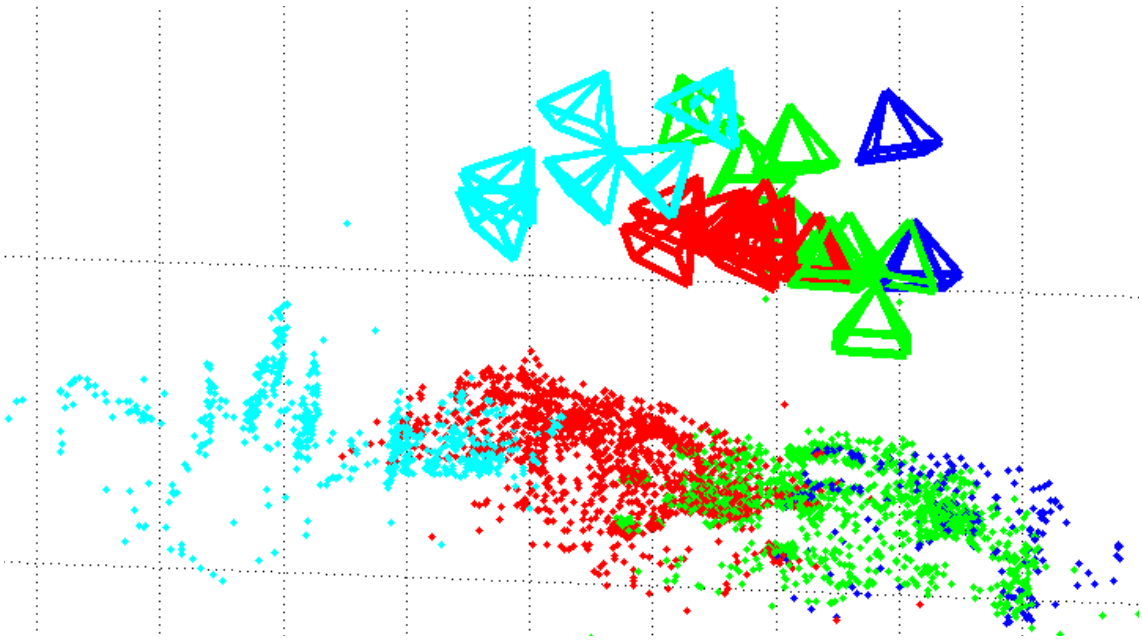


Figure 5.1: Scene *Carpet*. Highlighting participating clients. At this point in time, not all committed keyframes have been processed on the server. From left to right (cyan to blue): Clients A to D

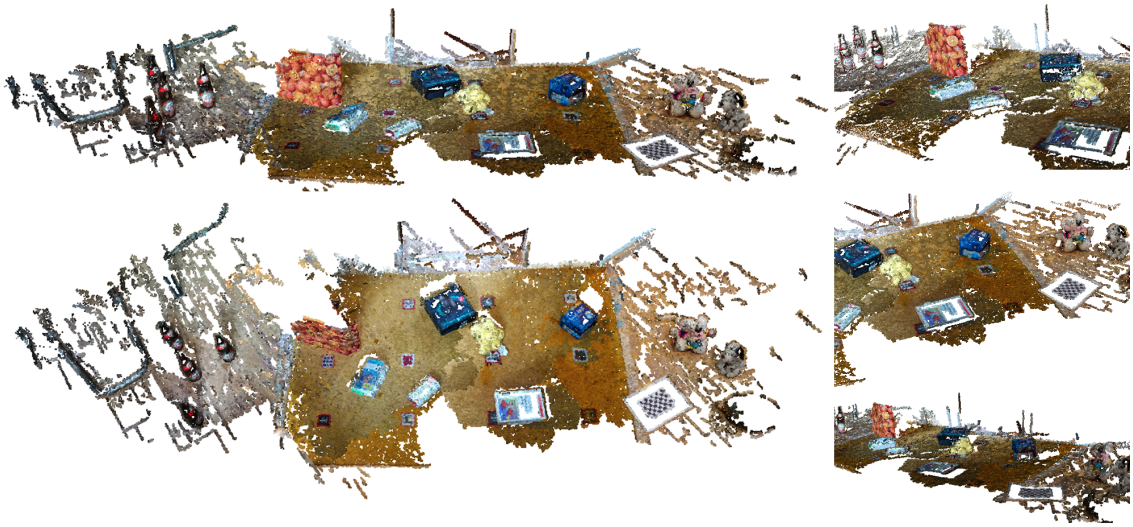


Figure 5.2: Scene *Carpet*. Reconstruction (densified with PVMS) from four merged clients.

(a) Client A

(b) Client B

(c) Client C

(d) Client D

Figure 5.3: Scene *Carpet*. **(a)** to **(d)** show some of the used input images on the left and client specific reconstructions (densified with PVMS) on the right and .

Client	Merge-Order	#committed Keyframes
A	3	9
B	0	12
C	1	9
D	2	7

Table 5.3: Scene *Carpet*. Clients and their share in the map.

Client	Merge-Order	#committed Keyframes
A	0	22
B	1	11
C	2	21

Table 5.4: Scene *Cardgame*. Clients and their participation.

5.2 Scene Cardgame

The *Cardgame* scene may be a possible real-world application. In this particular scenario three clients observe a game of *Magic the Gathering* with two players (static scene). In possible AR applications, cards could be augmented with 3D models or the whole environment could change during to the game. All clients act as *Fat Clients* in the three proposed operational modes. In all modes, the clients pull the same keyframes (related to them) from the server and try to improve or expand their local map accordingly. Table 5.4 shows the merge order and the number of committed keyframes. The clients are spatially arranged from left to right like B-A-C, with frame-sized overlapping areas, while this arrangement resembles a possible real-world scenario. Based on the reconstruction of the whole scene as shown in figure 5.2, information can easily be distributed amongst clients.

5.2.1 Client Mode: V1 - Keyframes

V1 only uses received keyframes to improve the local map. The keyframe is matched against the map and the pose is calculated. The used thresholds (RANSAC parametrisation for pose estimation) are rather strict, to avoid false positives. As described in Table 5.5, only five keyframes could be added to the current map, assuring that the calculated pose is almost correct. Therefore, we see an increased number of observations (nearly 2000 at t_5), which implies improved tracking quality. The number of 3D points remains the same, because the external keyframes are only matched against existing features and not used for triangulation of new areas. Skipped keyframes remain in the list for later addition.

This method performs good, in terms of adding more observations to known 3D points. The draw is, that the computational effort is to high for only adding new observations. Therefore we look forward to V2.

	t_0	t_1	t_2	t_3	t_4	t_5
#KF	23	24	25	26	27	28
#PT	1463	1463	1463	1463	1463	1463
#OB	13803	13878	14129	14468	14957	15452

Table 5.5: Scene *Cardgame*. Client A, V1. Changes over time. 3D-Points stay the same, new observations added.

5.2.2 Client Mode, V2 - Keyframes and related Poses

V2 operates very similarly to *V1*. The major difference is, that V2 relies on the server calculation. It assumes, that the server provides the correctly transformed poses for each particular client, which is the case. The computational effort is lower as in V1 and the amount of added keyframes and poses in the same amount of time is significant higher. Further, the problem of false positives, in terms of added keyframes with inaccurate or simply wrong pose estimates disappears. In this case, the server assures to provide the correct poses, by applying scale-, orientation-, origin- and offset-correction.

Table 5.6 shows the described behaviour. The increase in number of observations is around 9000. Again, the number of 3D points remains the same because no further triangulation is done between received and stored keyframes. We already can say, that V2 outperforms V1 because of the bigger versatility, with the single drawback of strongly relying on the provided poses. New keyframes are easier to add due to no or very weak dependency of points already included in the local map (No RANSAC for pose estimation needed). Figure 5.4 shows a clear difference between V1 and V2. V2 adds twice as much keyframes as V1. Important to note is that new keyframes are added close to the borders of the local scene, enabling the client to rapidly extend its map when moving further. The slight increase of 3D points in table 5.6 between t_0 and t_3, is caused by the asynchronous processing of the client and is not related to the received information.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
#KF	23	27	29	31	37	40	42	44	45	46	47
#PT	1465	1465	1467	1471	1471	1471	1471	1471	1471	1471	1471
#OB	13710	13911	15794	16627	20431	21713	21754	21799	22392	22405	22417

Table 5.6: Scene *Cardgame*. Client A, V2. Changes over time. 3D-Points stay the same, new Observations added.

5.2.3 Client Mode, V3

In terms of keyframes and poses, the result is similar to V2. Due to numerical deviations and iterative bundle adjustment, the results are not exactly the same. With the possibility of expanding the local map, V3 has more areas of application. For instance, one client stands still, consuming any kind of augmented content and an other client is expanding the combined map, in which both participate. Instantly the first client gets updates and enlarge its local map. If it decides to move, the tracking, can then be done seamlessly, with

little or no additional computational effort of its own SLAM-system. In this experiment the client has gone through its sequence and afterwards received the external poses and keyframes. However, we can turn it around, pre-load the external information at the beginning of the sequence, to observe that a majority of the points is generated by external frames for a specific area.

Table 5.7 shows increasing number of 3D points and observations, as new keyframes are added. Our implementation of V3 triangulates only once, as a keyframe arrives, and adds the keyframe and the new 3D points to the map. Another possible strategy is, to mark received keyframes for more triangulation approaches, until almost all detected natural features are used for 3D points, especially in edge regions of the local map. This would increase the amount of added 3D points, based on external and local keyframes. Further it would support the growth of the scene beyond the knowledge of a client, allowing faster tracking when leaving the known scene.

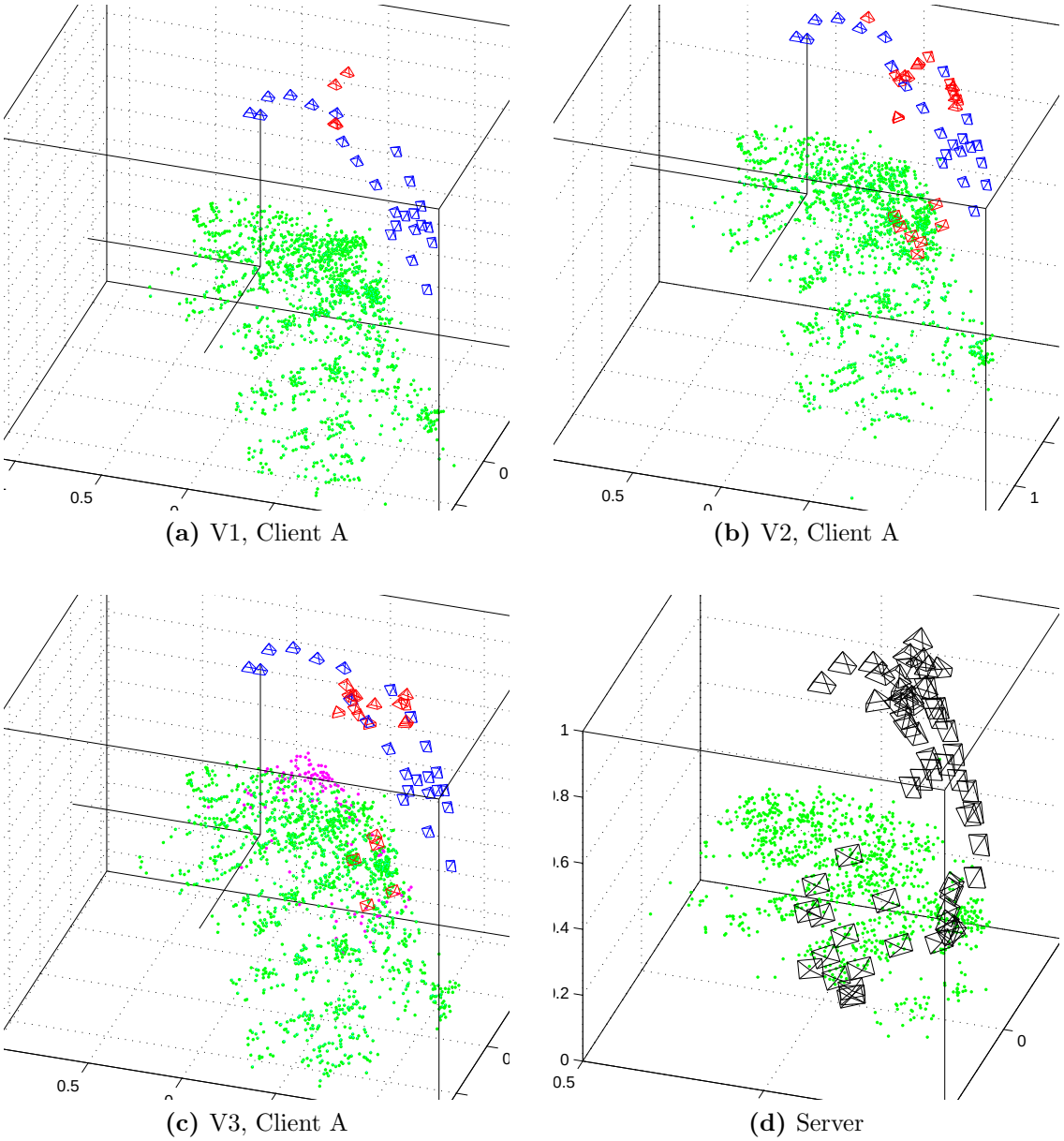
Comparing the three proposed methods, we can say V2 and V3 are most usefull where V1 could help clients with tracking difficulties and without the ability to operate in mode V2 or V3. Otherwise, V2 and V3 are superior to V1 with V3, at a bigger computational effort. Further, V3 extended the map as described, by adding more 3D points, without drastically influencing real-time performance. If keyframes are processed one at a time, V3 delivers acceptable real-time performance when performed on the Microsoft Surface Pro 3.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
#KF	24	28	34	37	39	41	42	43
#PT	1462	1462	1590	1590	1590	1590	1590	1590
#OB	15180	15180	20233	21010	21972	22466	22466	22466

Table 5.7: Scene *Cardgame*. Client A, V3. Changes over time. New 3D points added at t_2. Increased observations as seen in V2.

5.2.4 Server reconstruction

The *Cardgame* scene is reconstructed with 720p images acquired by three clients. The reconstruction is in grayscale because the clients committed only grayscale images to save bandwith. Figure 5.5 and 5.6 show the client and server reconstructions with some related input images. Further, our server system was able to detect correct image overlaps and to calculate valid merges. Bundle adjustment is only applied to the individual client parts and not on the merged map. Additionally, figure 5.6 shows reduced holes of client C, with information of client A. Visualising the growth over time, shows that external keyframes integrate very well into the local map and help to improve the tracking. Especially, figure 5.4 image (c) shows new 3D points in magenta, which extend the scene and will speed up tracking, if the client moves into this area.



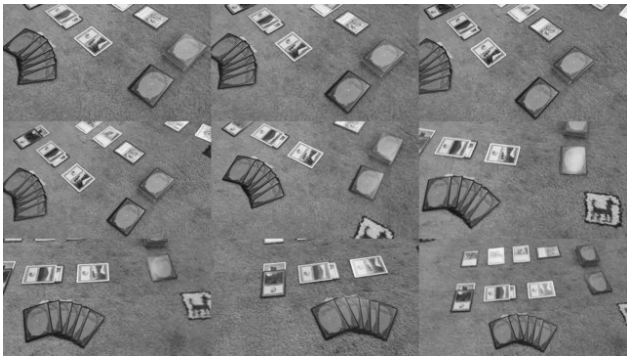
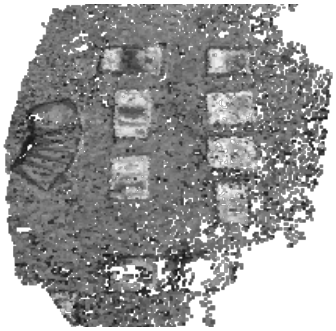
(a) V1, Client A **(b)** V2, Client A

(c) V3, Client A **(d)** Server

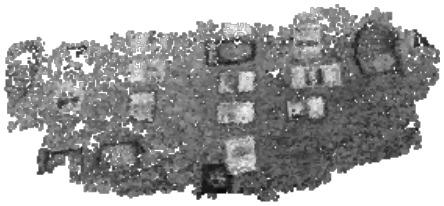
Figure 5.4: Autonomous client mode comparison. Scene *Cardgame*. Local poses in *blue*, received keyframes/pose in *red*. **(a)** and **(b)** received the same keyframes from the server. V1 estimated the poses and V2 used the server estimate. **(c)** shows the server map.

5.3 Scene Office

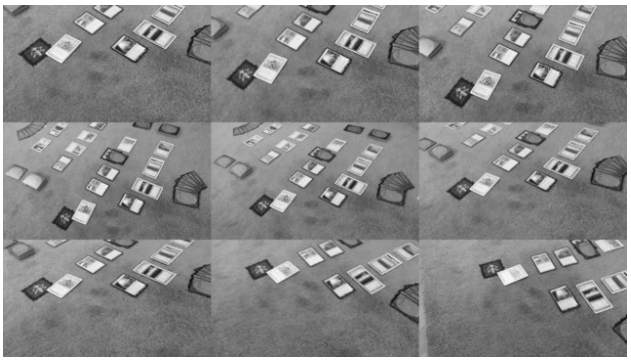
The *Office* scene is our third test environment and mainly used as showcase. Clients operate as autonomous clients in V3. The clients are spatial placed along a U-shaped desk in an order like C-A-B with A as initial client. Figure 5.7 compares the local client to the server reconstruction and Figure 5.8 shows the growth of one client over time. Figure 5.9 shows the reconstruction including some committed keyframes and Figure 5.10

(a) Client C

(b) Client A

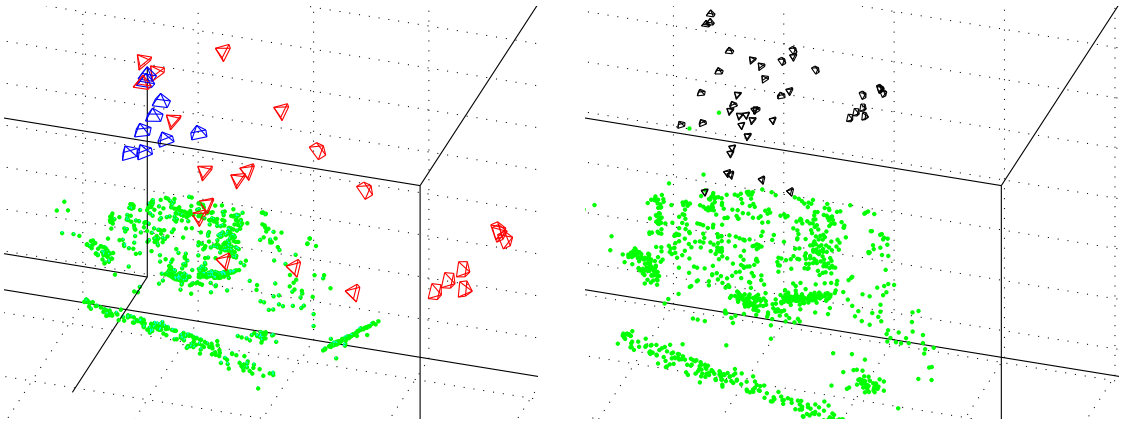



(c) Client B

Figure 5.5: Scene *Cardgame*. **(a)** to **(c)** some of the used input images on the left and client specific reconstructions (densified with PVMS) on the right.

Figure 5.6: Scene *Cardgame*. Server reconstruction (densified with PVMS) of combined clients.

compares the trajectory recorded with an external tracker system, to the resulting server reconstruction. Our reconstructed trajectory based on keyframes is very similar to the recorded one with deviations mainly with respect to scale.



(a) Client V3 **(b)** Server

Figure 5.7: Scene *Office*. **(a)** shows the client and including external poses. **(b)** shows the server reconstruction. At this point in time the client has no processed all received keyframes. The differences in scale regarding to the 3D points, is due to different systems with different initializations e.g. initial distance to the first triangulated point.

5.4 Application Cases

In this section we describe various application cases and discuss one in detail, for which our system could serve as an enabling solution. This cases may operate with annotations,

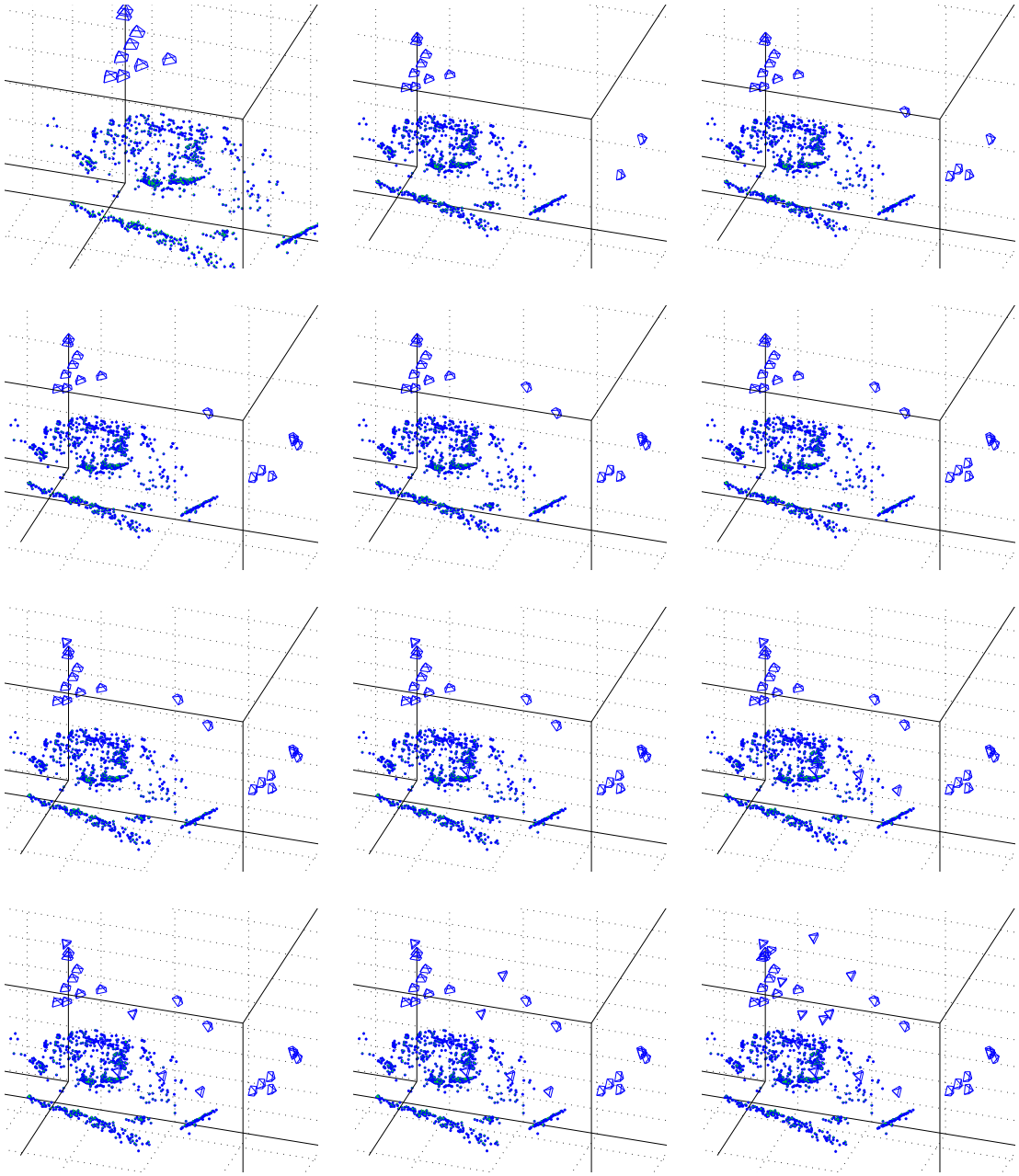


Figure 5.8: Scene *Office*. Fat client A, V3. Map growth over time. Starting at top left with no external keyframes until bottom right, where almost all external keyframes have been added.

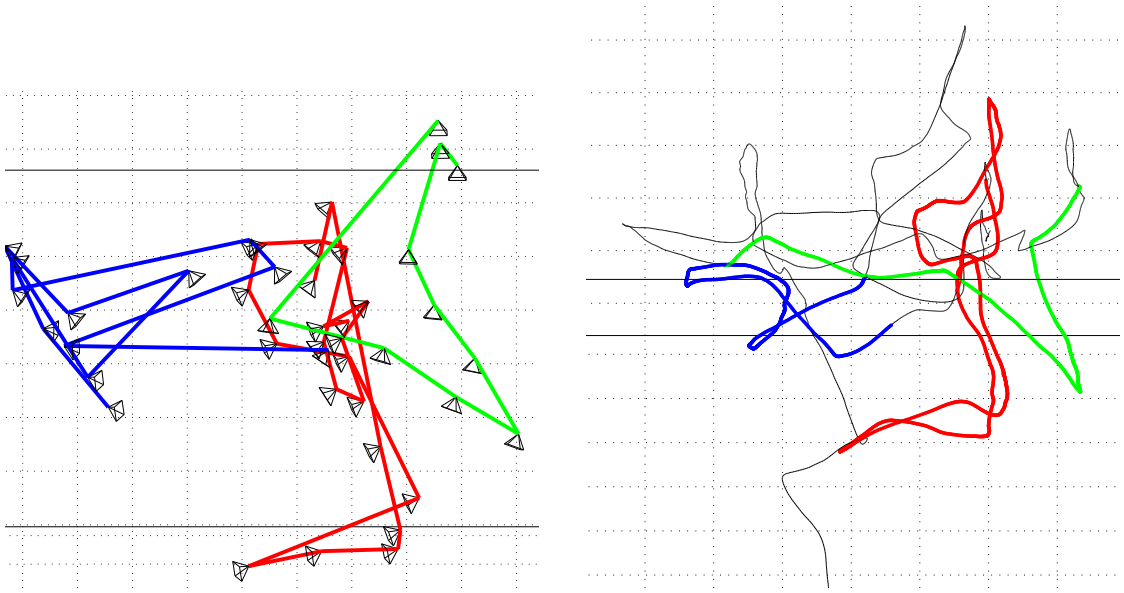
(a) Client A

(b) Client B

(c) Client C

(d) Merged map

Figure 5.9: Scene *Office*. Reconstructions (densified with PVMS) and keyframes. **(a)** to **(c)** show clients A, B and C where spatial arrangement is B-A-C. Left shows some of the commuted keyframes and right the reconstruction. **(d)** shows the merged map of all three clients from two different views (front and above).



(a) Our **(b)** External tracker

Figure 5.10: Scene *Office*. Camera pose comparison. **(a)** shows the reconstructed server trajectory based on committed client keyframes and **(b)** shows the trajectory of an external tracker. Comparing (a) and (c), shows very similar trajectory. Small deviations are related to the fact, that (b) was recorded as a single recording and (a) represents parts of the recording committed by three different clients. *Red*, *green* and *blue* represent clients. The green line in (a) deviates on the left side from the green line in (b), because of the loop-playback used.

which allow to mark specific points in the collaborative map. Annotations are created by clients and mapped into the collaborative map, for a possible update of all clients. The server is used to transform annotations taken by one client, into valid 3D spaces of other clients.

Case (i) describes a typical household situation: one recognizes in the morning, that the milk is empty, but he is in a hurry to catch his train. When standing in front of the fridge with an empty bottle of milk, it is possible to place a virtual model of the bottle near the virtual fridge, so that the flatmate, once he or she gets up, recognizes it and buys a new one.

Case (ii) takes place in a (partially) empty flat, where a couple is currently moving in, but they are not sure how to place their new furniture, or even which furniture to buy. Software like *Sweet Home 3D*[1] is a home furnishing tool, which allows to define the space and to place objects within, but without any reconstruction or immersion, and one has to do all measurements for walls, windows and doors on his own. Using an approach like our proposed system as a basis allows them to register the whole furniture to the collaborative map and deliver it to all clients. One advantage is, that the registration is against the collaborative map and not against one client. Therefore, the server can

[1] `http://www.sweethome3d.com/`

transform annotations to all needs, because clients are in the same space. Ones a client is localized within the map, the map could be enlarged or refined and furniture would be correctly visualized.

Case (iii) is like the other cases, but to a bigger scale. It is often difficult to find the correct corridor in an DIY-store, because of the high shelves. Such a store could have its own (non-perfect) reconstruction. When entering the store, our client would be merged into the existing reconstruction, and information about what products are in which corridor, would be easily accessible. The user could then e.g. filter the information for different bolts and screws. On the clients visualization, we would highlight the related corridors and on a more complex application, the user could be guided to shelves of the concerned items. Further, screws of specific sizes could be annotated to decrease the time needed to find the searched ones.

All cases have in common, to share information between clients, with respect to the collaborative map and with respect to a certain client. Information is added by clients, for other clients. Because of the collaborative map, the server can easily translate between different clients, and compensate for different client implementations. The following section will discuss case (i) in more detail.

5.4.1 AR scenario - The forgotten milk

As previously described, the task is to annotate the fridge or any space close by, and to visualize the annotation between all clients. In particular, we use two clients on a usual kitchen scene. Figure 5.11 shows the scene including the rendering of the milk carton. An annotation can be created by a client and committed to the server, or can be directly created by the server. In both cases clients will receive updates about annotations. Transformations between fat clients and the server with high accuracy in placement and orientation turn out to be difficult. The problem is mainly regarded to the individual bundle adjustment algorithms used by clients and the server.

The annotation is more accurate at the beginning of the sequence when only a few keyframes are captured and will deviate in position and orientation when more keyframes are added on. The client-side, as well the server-side, BA algorithms move points and poses for optimization and change the 3D structure of each reconstruction. Annotations remain on the same 3D coordinates, but the underlining 3D reconstruction changes. This effect is shown in the second row of figure 5.11, where the milk carton is on the left side of the cup for client A and almost on the right side for client B. Figure 5.12 shows the deviation between client A and the server and figure 5.13 for client B and the server. The deviation remains small for the first few keyframes and grows over time. Due to individual systems with different algorithms and loose server dependencies of our proposed system, the accuracy for use cases like annotations is limited. The described problem affects only client-server transformation, which need to remain accurate over time e.g. annotating a cup of tea. The collaborative reconstruction remains valid, as shown in figure 5.14.

Figure 5.11: Annotation in two client spaces. The left column shows representative keyframes of client A and the right column of client B. The annotation is registered against the merged map and updated to clients. Comparing the position of the milk bottle, we observe inaccurate placements caused by deviating server and client systems.

5.4.2 Discussion

With each new keyframe the optimization slightly changes the reconstruction and the positions and orientations of the poses, which will lead to bigger deviations. If we tighten the client-server dependencies, we can eliminate this behaviour by applying one of three possible methods, namely *pose alignment and 3D point update*, *aligning of 3D structures* and *anchor points to fix pose-3D-point distances*.

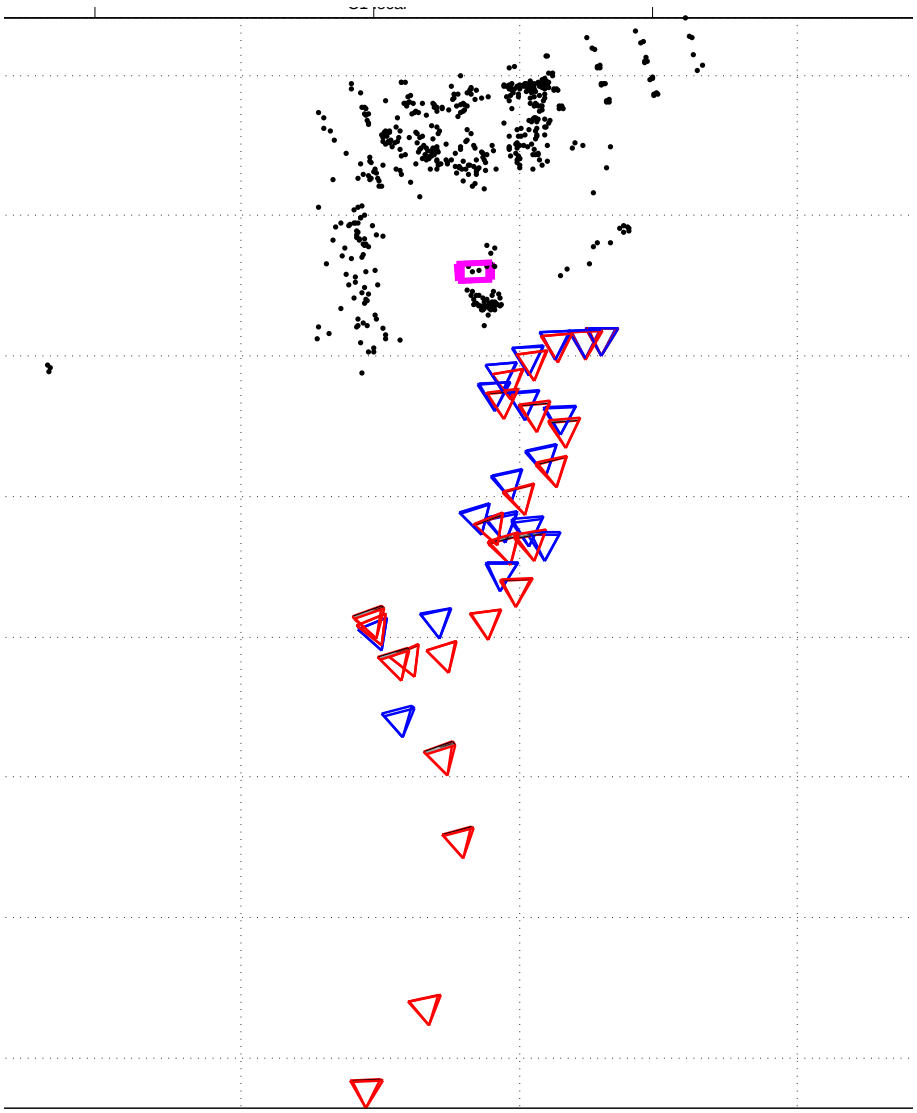


Figure 5.12: Local poses of client A are shown in red. The corresponding server poses transformed into the local space of client A are shown in blue. The annotation is shown in magenta and the local 3D points in black. The movement starts on the top right towards the lower end of the image. Within the first few keyframes the client and server poses match with little to no error. After some bundle adjustment iterations we can see, that the systems are deviating, and the poses are no more aligned. This also holds for the 3D points. The annotated 3D position is transformed correct, but the 3D structure of client A has moved over time. *Note*: The annotation should be at the black dots below its current position.

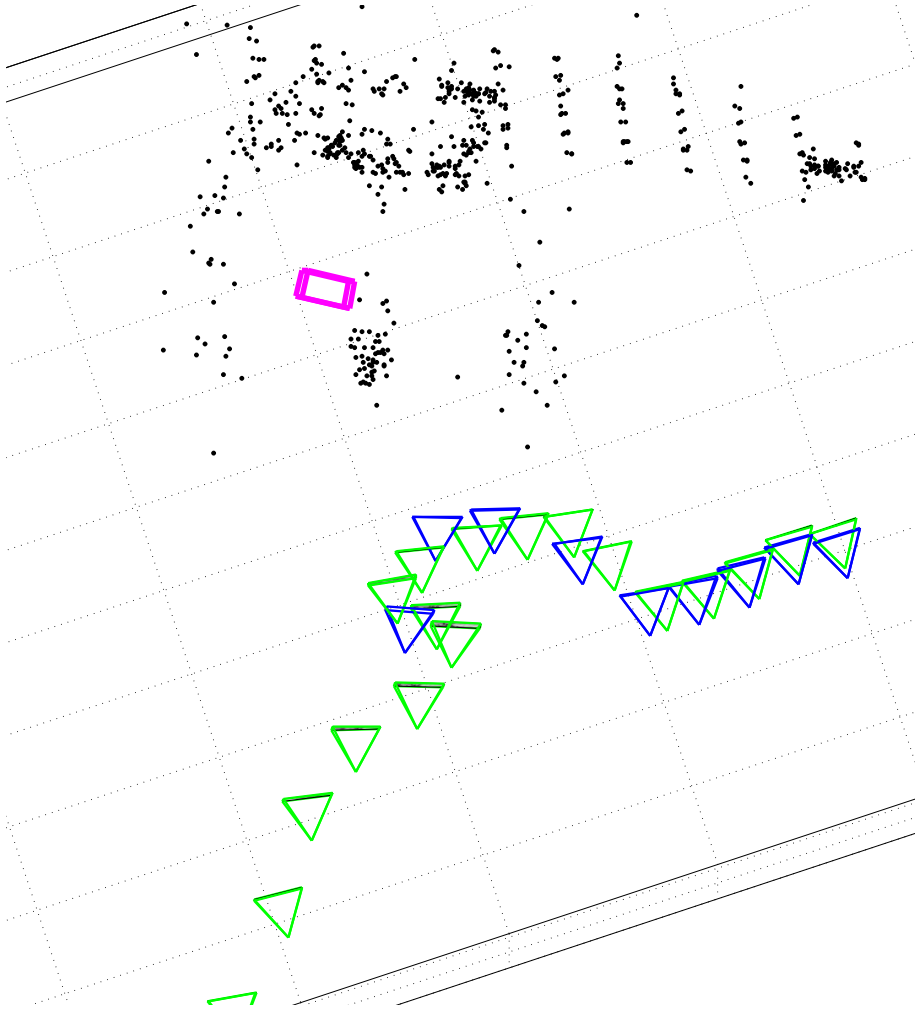
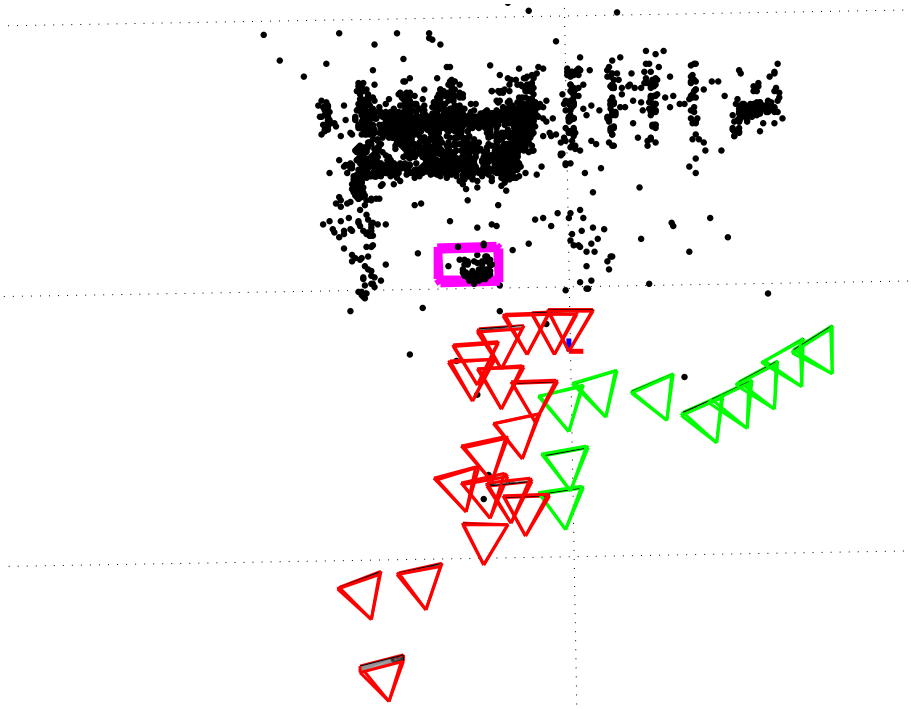


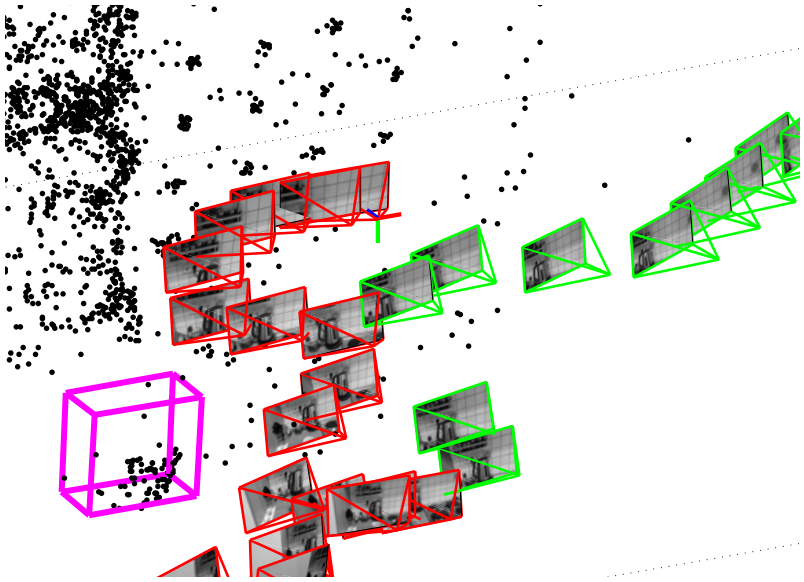
Figure 5.13: Local poses of client B are shown in green and the corresponding server poses transformed into the local space of client B are shown in blue. The annotation is shown in magenta and the local 3D points of client B are shown in black. We see fewer server poses (blue), because at this point in time not all server poses have been processed. The client movement starts on the far right towards the left lower side of the image. With each added keyframe the deviation between server and client poses grew, leading to an in-precise placement of the annotation. *Note:* The annotation should be placed left to the points below the annotation. As described, the deviation is manly caused by individoual systems with different algorithms e.g. bundle adjustment

5.4.2.1 Pose alignment and 3D point update

The idea behind this method is, that the server corrects client poses and the client corrects local 3D points related to these poses. Due to the bigger capabilities and more precise algorithms of the server, the poses calculated by the server are more accurate. Therefore, the server could prepare pose updates, which notify clients about corrected pose positions and orientations. In this case, the client has to re-position and re-orient local poses with the received ones. If local poses are moved affected 3D points become inaccurate, because



(a) top view



(b) side view

Figure 5.14: Annotation registered in the merged map. Client A with red poses and client B with green poses have collaboratively created the merged map by committing their keyframes to the server. 3D points are in black and the annotation is shown in magenta. **(a)** shows the top view and **(b)** the view from the side.

of the new pose position. Those 3D points have to be corrected in position. We can assume, based on the bundle adjustment and the more accurate server poses, that the influence of the optimization would be reduced, because a new keyframe would have more accurate predecessors, which it could rely on. Advantages of this method are more accurate client poses and negligible effort for the server. Disadvantages include increased communication effort (pose update after each bundle adjustment) and increased computational effort for the client, which could affect real-time performance, if on every update a majority of the points and the poses had to be corrected. Changes to the majority of poses seem to be uncommon, assuming that the server computation is correct and only new client poses and 3D points have to be fixed. The Horn algorithm can be used to perform this alignment, but it will fail when poses are positioned co-linear, which creates a further disadvantage.

5.4.2.2 Aligning of 3D structures

This approach tries to align two 3D point clouds. After alignment, the reconstructions of the server and the client are identically placed and therefore annotations could be placed more precise. In particular, the server would serve as reference and send the reconstruction or parts of it to the client. Afterwards, the client tries to align the local reconstruction with the received one. Iterative closest point algorithms, as shown by Rusinkiewicz *et al.* [46] can be used to perform the alignment. This method should be more precise, but also use a high bandwidth based on the amount of 3D points needed for a good alignment. ICP algorithms can be computational expensive and seriously affect real-time performance on the client. Further, both considered point clouds are homogeneous and do not guarantee a successful alignment (basicaly, there are no one-to-one 3D point correspondences between those point clouds). The deviations between two 3D point cloud representing the same environment, can be to large causing ICP algorithms to fail or deliver bad results.

5.4.2.3 Anchor points to fix client-server drift

The basic idea behind anchor points is to provide a set of corresponding 2D and 3D points, which will remain constant while the bundle adjustment performs the optimization. This point set can be determined and provided by the server, allowing the server to strongly influence the client map. Of coarse, the provided 3D points are chosen from the merged map, forcing clients to align their poses and points with the server. Ventura *et al.* show, how to use such anchor points for alignment between server and client [59]. For good results, it is necessary that the server bundle adjustment uses the same anchor-points, or provided ones are updated when changed. We can already guess, that this method has low impact on client side real-time performance, because it basically adds constraints to the bundle adjustment. Of course, an increased amount of data has to be transmitted.

In our case, the *anchor-points* method looks the most promising, because it has low impact on the real-time performance, when compared to the other methods. All methods

seem to solve the client-server deviation problem, and will need tighter dependencies to the server while using more bandwidth.

5.5 Discussion

All tested indoor environments worked very well. If an image overlap leads to an valid epipolar geometry, the merge can correctly be calculated. Refreshing the stored maps with new client information did not introduce errors. In terms of performance, our presented system is very parallelizable. Each clients SfM pipeline, the image overlap detection and the map management can be split up into individual threads. Further, the search strategy used in the overlap detection, could be replaced with an tree search based one, which would allow to parallelize the algorithm itself and additionally, information of client movement or position could be taken into account, to speed up the search process. According to the overlap detection, the verification of a single image pair takes a couple of milliseconds.

Based on the strict merge validation, merge errors could be avoided. When lowering the thresholds. For instance of a valid epipolar geometry, it is possible to generate false merge information. Otherwise our approach is very robust on the tested scenes. Problems may occur for clients, which use weak features for image matching on images with repetitive patterns.

Our annotation extension has room for improvements as mentioned in previous sections. To transform annotations to the merge-map space, the first clients camera is used. It may be better to use the closest camera for the transformation, if the annotation is far away from the first camera to increase accuracy. Annotations made in the merged map space, are accurate in position, but will face deviation problems when different clients are used. The system is capable to handle multiple annotations per client and to provide updates for other clients.

6 Conclusion

6.1 Conclusion and future work

We presented a method to combine various clients, which operate within the same environment to generate a combined map. Clients can collaboratively generate maps and navigate within the same context, which allows for sharing annotations for AR related applications. The created maps can be combined to even bigger ones, if clients close the gaps between existing maps. Further, we do not restrict to specific clients and allow the server system to handle different clients and compensate for deviations among their local systems. Combined maps can be managed and expanded by new or by already participating clients. Our server system is capable to perform transformations, to combine clients, to generate a merged map. Additionally, the server can react to different client systems and compensate for deviating bundle adjustment implementations and cases like different orientations.

Three operational modes for a specific SLAM client were presented, namely V1 (keyframes only), V2 (keyframes and poses) and V3 (keyframes and poses with triangulation). This allowed different usage scenarios for improving the local system. Our autonomous client decides on his own, how to add external server information. V2 and V3 seem to be more useful in AR specific applications. V1 could improve the tracking of one client, which is not capable to operate in mode V2 or V3. Our results show a pleasant performance among various indoor scenes, with V3 giving the most benefit.

Our contributions of this work are: (i) A server system to collect and combine data from various clients to large maps and how to distribute the results in a very loose way, and (ii) how to use the received data in local SLAM systems to improve the performance in terms of mapping and tracking, without noticeable affecting their real-time constraints.

The next steps of our work are the design of an AR application to show context sharing over multiple clients. Further, to try different strategies for client updates, namely in which order clients would receive their specific updates. Also,

additional strategies for detecting image overlaps between clients are considered e.g. tree based approaches, which consider additional information, like position data. Large scale tests, which go beyond the computational capabilities of a desktop pc, would be interesting, as well as outdoor environments.

This work shows, that there is a plausible need for such a system to allow different clients to operate in the same AR environment. We can imagine, from an applications point of view, to create an operating system for different independent AR applications. For instance, one has an AR related app for furniture placement. While placing the furniture, a map of the room would be generated. Instead of loosing the map after shutting down, the map would reside on the server system. After turning off the furniture app, one would start an AR environment-related game. Instantly, the client would get updates and the whole processing capability of the device could go into the game, without the need of initially creating a map. If ones friend would like to join the game, he also would get the updates and both would reside within the same AR context and continual actualize each other.

Further, the server system could support different modes, in terms what kind of updates to provide for different kinds applications like, keyframes, keyframes and poses, transformation to move the client into the combined-map-coordinate system, the whole map including 3D points or even 3D reconstructions for visualization purposes. Clearly, there is a need for such a system in AR related applications.

Bibliography

[1] Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. (2011). Building Rome in a Day. *Commun. ACM*, 54(10):105–112. (page 14)

[2] Agarwal, S., Mierle, K., and Others (2014). Ceres solver. (page 10)

[3] Agarwal, S., Snavely, N., Seitz, S. M., and Szeliski, R. (2010). Bundle adjustment in the large. In *Proceedings of the 11th European Conference on Computer Vision: Part II*, ECCV'10, pages 29–42, Berlin, Heidelberg. Springer-Verlag. (page 10)

[4] Akbarzadeh, A., m. Frahm, J., Mordohai, P., Engels, C., Gallup, D., Merrell, P., Phelps, M., Sinha, S., Talton, B., Wang, L., Yang, Q., Stewénius, H., Yang, R., Welch, G., Towles, H., Nistér, D., and Pollefeys, M. (2006). Towards urban 3d reconstruction from video. In *in 3DPVT*, pages 1–8. (page 14)

[5] Angeli, A., Filliat, D., Doncieux, S., and Meyer, J. (2008). A fast and incremental method for loop-closure detection using bags of visual words,â conditionally accpeted for publication in. *IEEE Transactions On Robotics, Special Issue on Visual SLAM*. (page 15)

[6] Armstrong, M., Zisserman, A., and Beardsley, P. A. (1994). Euclidean reconstruction from uncalibrated images. In *British Machine Vision Conference*, pages 509–518. (page 13)

[7] Baillard, C. and Zisserman, A. (1999). Automatic reconstruction of piecewise planar models from multiple views. In *CVPR*, pages 2559–2565. IEEE Computer Society. (page 13)

[8] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of Optical Flow Techniques. *Int. J. Comput. Vision*, 12(1):43–77. (page 31)

[9] Beardsley, P., Torr, P., and Zisserman, A. (1995). 3d model acquisition from extended image sequences. (page 13)

[10] Bouguet, J. (2000). Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*. (page 31)

[11] Bujnak, M., Kukelova, Z., and Pajdla, T. (2008). A general solution to the p4p problem for camera with unknown focal length. (page 8)

[12] C., H., M., K., M., R., A., W., S., K., H., B., and Reitmayr, G. (2012). Online Feedback for Structure-from-Motion Image Acquisition. (page 15, 19)

[13] Castle, R. O., Klein, G., and Murray, D. W. (2008). Video-rate Localization in Multiple Maps for Wearable Augmented Reality. In *Proc 12th IEEE Int Symp on Wearable Computers, Pittsburgh PA, Sept 28 - Oct 1, 2008*, pages 15–22. (page 17)

[14] Davison, A. J. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1403–, Washington, DC, USA. IEEE Computer Society. (page 15)

[15] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067. (page 16)

[16] Dementhon, D. F. and Davis, L. S. (1995). Model-based object pose in 25 lines of code. *Int. J. Comput. Vision*, 15(1-2):123–141. (page 8)

[17] Deriche, R., Kornprobst, P., and Aubert, G. (1995). Optical-Flow Estimation while Preserving Its Discontinuities: A Variational Approach. In Li, S. Z., Mital, D. P., Teoh, E. K., and 0001, H. W., editors, *ACCV*, volume 1035 of *Lecture Notes in Computer Science*, pages 71–80. Springer. (page 31)

[18] Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-whyte, H. F., and Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17:229–241. (page 15)

[19] Endres, F., Hess, J., Sturm, J., Cremers, D., and Burgard, W. (2013). 3D Mapping with an RGB-D Camera. *IEEE Transactions on Robotics (T-RO)*, 30(1):177–187. (page 16)

[20] Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. (page 17)

[21] Engels, C., Stewénius, H., and Nistér, D. (2006). Bundle adjustment rules. In *In Photogrammetric Computer Vision*. (page 10)

[22] Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6):381–395. (page 23)

[23] Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R. (2010). Towards Internet-scale Multi-view Stereo. In *CVPR*. (page 20)

[24] Furukawa, Y. and Ponce, J. (2010). Accurate, Dense, and Robust Multi-View Stereopsis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376. (page 20)

[25] Golub, G. and Reinsch, C. E. (1970). Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420. (page 7)

[26] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition. (page 5, 6)

[27] Hedborg, J., Felsberg, M., Hedborg, J., and Felsberg, M. (2013). Fast iterative five point relative pose estimation. (page 7)

[28] Ho, K. L. and Newman, P. (2007). Detecting loop closure with scene sequences. *Int. J. Comput. Vision*, 74(3):261–286. (page 15)

[29] Hook, A., Fite-Georgel, P., Meisnieks, M., Maes, A., Gardeya, M., and Naimark, L. (2014). Generation and sharing coordinate system between users on mobile. (page 18)

[30] Horn, B. K. P., Hilden, H., and Negahdaripour, S. (1988). Closed-Form Solution of Absolute Orientation using Orthonormal Matrices. *JOURNAL OF THE OPTICAL SOCIETY AMERICA*, 5(7):1127–1135. (page 9, 23)

[31] Irschara, A., Zach, C., and Bischof, H. (2007). Towards Wiki-based Dense City Modeling. In *ICCV*, pages 1–8. IEEE. (page 14)

[32] Irschara, A. ; Hoppe, C. . B. H. . K. S. (2011). Efficient structure from motion with weak position and orientation priors. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 21–28, Nara, Japan. (page 15)

[33] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. (2011). Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. ACM Symposium on User Interface Software and Technology. (page 16)

[34] Klein, G. and Murray, D. (2007). Parallel Tracking and Mapping for Small AR Workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan. (page 16)

[35] Klein, G. and Murray, D. (2009). Parallel Tracking and Mapping on a Camera Phone. In *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando. (page 16)

[36] Knapp, A. W. (2002). *Lie Groups Beyond an Introduction.* Birkhäuser (2002) ISBN 0-8176-4259-5. (page 11)

[37] L. Riazuelo, Javier Civera, J. M. (2014). C2TAM: A Cloud framework for cooperative tracking and mapping. In *Robotics and Autonomous Systems, Volume 62, Issue 4, Pages 401â-413, April 2014.* (page 17, 18)

[38] Lourakis, M. I. A. and Argyros, A. A. (2009). SBA: a software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, pages 1–30. (page 10, 14)

[39] Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA. IEEE Computer Society. (page 1, 22)

[40] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada. AAAI. (page 15)

[41] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). DTAM: Dense Tracking and Mapping in Real-Time. (page 16)

[42] Nistér, D. (2004). An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):756–777. (page 7, 19)

[43] Nistér, D. and Stewénius, H. (2006). A Minimal solution to the generalized 3-point pose problem. *Journal of Mathematical Imaging and Vision*. (page 8, 19, 25)

[44] Ogg, A. (1991). Review: Igor frenkel, james lepowsky and arne meurman, vertex operator algebras and the monster. *Bull. Amer. Math. Soc. (N.S.)*, 25(2):425–432. (page 11)

[45] Pollefeys, M., Nistér, D., Frahm, J. M., Akbarzadeh, A., Mordohai, P., Clipp, B., Engels, C., Gallup, D., Kim, S. J., Merrell, P., Salmi, C., Sinha, S., Talton, B., Wang, L., Yang, Q., Stewénius, H., Yang, R., Welch, G., and Towles, H. (2008). Detailed Real-Time Urban 3D Reconstruction from Video. *Int. J. Comput. Vision*, 78(2-3):143–167. (page 14)

[46] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*. (page 58)

[47] Serre, J.-P. (2001). Complex semisimple lie algebras. (page 11)

[48] Sinha, S. N., Jan-michael Frahm, Pollefeys, M., and Genc, Y. (2006). GPU-based Video Feature Tracking and Matching. Technical report, In Workshop on Edge Computing Using New Commodity Architectures. (page 19, 22)

[49] Smith, R., Self, M., and Cheeseman, P. (1990). Autonomous robot vehicles. chapter Estimating Uncertain Spatial Relationships in Robotics, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA. (page 15)

[50] Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo Tourism: Exploring Photo Collections in 3D. *ACM Trans. Graph.*, 25(3):835–846. (page 13)

[51] Snavely, N., Seitz, S. M., and Szeliski, R. (2008). Modeling the World from Internet Photo Collections. *Int. J. Comput. Vision*, 80(2):189–210. (page 13)

[52] Stewénius, H., Engels, C., and Nistér, D. (2006). Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:284–294. (page 7, 19)

[53] Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2011). Real-time monocular slam: Why filter? (page 15)

[54] Strecha, C., Pylvänäinen, T., and Fua, P. (2010). Dynamic and scalable large scale image reconstruction. In *CVPR*, pages 406–413. IEEE. (page 14)

[55] Sun, D., Roth, S., and Black, M. J. (2010). Secrets of optical flow estimation and their principles. (page 31)

[56] Sweeney, C. (2013). Improved Outdoor Augmented Reality through Globalization. In *Doctoral Consortium, ISMAR'13*. (page 18)

[57] Tan, W., Liu, H., Dong, Z., Zhang, G., and Bao, H. (2013). Robust monocular SLAM in dynamic environments. In *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2013, Adelaide, Australia, October 1-4, 2013*, pages 209–218. IEEE Computer Society. (page 16)

[58] Turk, M., Hollerer, T., Ventura, J., Sweeney, C., and Gauglitz, S. (2014). Model Estimation and Selection towards Unconstrained Real-Time Tracking and Mapping. *IEEE Transactions on Visualization and Computer Graphics*, 20(6):825–838. (page 17)

[59] Ventura, J., Arth, C., Reitmayr, G., and Schmalstieg, D. (2014). Global Localization from Monocular SLAM on a Mobile Phone. *IEEE Transactions on Visualization and Computer Graphics*. (page 17, 58)

[60] Vergauwen, M. and Van Gool, L. (2006). Web-based 3D Reconstruction Service. *Mach. Vision Appl.*, 17(6):411–426. (page 14)

[61] Wendel, A., Maurer, M., Graber, G., Thomas, P., and Bischof, H. (2012). Dense reconstruction on-the-fly. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 1450–1457, Washington, DC, USA. IEEE Computer Society. (page 16)

[62] Zou, D. and Tan, P. (2013). CoSLAM: Collaborative Visual SLAM in Dynamic Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):354–366. (page 17, 18)