Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Localization and mapping of interior and shared augmented reality

Bachelor thesis

2019

Matej Sládek

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Localization and mapping of interior and shared augmented reality

Bachelor thesis

Študijný program: Informatika
Študijný odbor: Informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Vladimír Boža, PhD.

2019

Matej Sládek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Matej Sládek
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Localization and mapping of interior and shared augmented reality
Lokalizácia a mapovanie interiéru a zdieľaná rozšírená realita v ňom

Anotácia: Cieľom bakalárskej práce je vytvorenie kompletnej android aplikácie, ktorá umožňuje mapovanie a lokalizovanie sa v prostredí a následne umožňuje užívateľom dopĺňať objekty do tohto prostredia a zdieľat ich s ostatnými užívateľmi na spôsob sociálnych sietí. Sučasťou bakalárskej práce je identifikovanie najväčších problémov kvality a presnosti online 3d rekonštrukcie na zariadení a ich riešenie pomocou offline 3d rekonštrukcie na serveri, s ktorým aplikácia bude komunikovať.

Vedúci: Mgr. Vladimír Boža, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 31.10.2018

Dátum schválenia: 06.11.2018

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Za pomoci najmodernejších, voľne dostupných open-source SLAM a SfM technológií, sme vytvorili plne funkčný systém na zdieľanú rozšírenú realitu v interiéri pre viacerých hráčov. Udržiavame si lokácie všetkých užívateľov a obsah nimi vytvorený v globálnej mape na serveri a zdieľame ho so všetkými klientmi v reálnom čase. Toto tvorí základ pre každú reálnu aplikáciu využívajúcu rozšírenú realitu. Tiež hodnotíme odozvu a presnosť a identifikujeme hlavné problémy tohto systému a navrhujeme miesta na zlepšenie. Takisto poskytujeme 3D vizualizér, ktorý umožňuje zobrazovať globálnu mapu, pozície užívateľov v reálnom čase a obsah vytvorený užívateľmi.

Kľúčové slová: rozšírená realita, lokalizácia, mapovanie, SLAM, SfM, 3D

Abstract

Using state-of-the-art free and open-sourced SLAM and SfM we build fully functional multiplayer shared augmented reality indoor experience. We keep track of all user locations and user-generated content in one global map on the server shared with clients in real-time. This lays the foundation for any real application of shared augmented reality. Also we evaluate latency and accuracy and identify main bottlenecks and directions for improvements of such a system together with providing the number of debug tools like 3D visualizer of the global map detected features and real-time position of users and the content.

Keywords : augmented reality, localization, mapping, SLAM, SfM, 3D

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Objectives

In this bachelor thesis, we create an Android app which is able to provide shared augmented reality experience, that means it can localize our device in an environment with the precision of a few centimeters, render augmented reality(AR) objects in app view and be able to add objects to the environment and share them with other users. Then we investigate what the biggest issues and problems are and we analyze them.

Applications

The use of this system is in AR applications. In the game industry, there are already multiple mobile games which you can play in your environment and this would permit to make these games multiplayer. Ikea developed the application[5] to decorate your room with Ikea furniture, so you can run the application and see how would your room look like and then buy their stuff, this project would allow to save progress and then load it and share it with other users. AR Mole is a mobile game which shows moles on the screen as they would be in the real world and they disappear as you hit them[7]. There are multiple other areas like interactive museums, virtual boards, and others.

Problems

Localization and mapping is an essential problem for multiple applications. Localization is useful for knowing where we are and which virtual object should be displayed. Localization gives us a position on the map. The map is a list of positions of feature points and objects in the environment. Mapping is a process of adding new features and adjusting old features as we explore the environment. Mapping is important in order to do localization in this map and to understand the new unknown environment and store virtual objects in the map. The problem of simultaneous localization and mapping is called SLAM. SLAM continues to be under active

research for the last 2 decades.[6] SLAM appears to be a chicken-egg problem because localization depends on mapping and vice versa. One of the first monocular camera SLAM was MonoSLAM[4] which was introduced in 2003.

There are libraries for SfM and localization on the server and libraries for SLAM on the client. The problem is that there is no end-to-end solution to share the global map and have a consistent experience for multiple users on the mobile device. We try to fill this gap. Our work is to combine already established technologies as SfM on the server and SLAM on the client and to create an end-to-end solution including communication between these systems and moving responsibility of sharing the content from client to server to enable a higher number of anchors without decreasing overall performance.

Contribution

In this thesis, we build more robust SLAM system for AR content in the interior. We combine SLAM on the mobile phone and SLAM on the server. SLAM on the mobile phone is less accurate, have an only local map, but we have instant information about the position. SfM on the server has a global map and store AR objects and can do more robust optimization, so our localization is more accurate. Moreover we moved storing anchors from client to server which enabled us to remove the common limitation of similar systems that the increasing number of anchors decreases performance. We take technologies for SLAM on the client and SfM on the server and combine them and create an end-to-end solution for AR mobile applications.

Organization

In chapter 2 we get into the theory of localization and mapping and we describe basic principles of SLAM. In chapter 3 we describe an overview of our solution. In chapter 4 we get into sharing a pose between local and global map. In chapter 5 we describe the process of building a map. In chapter 6 we describe our software design. In chapter 7 we discuss the evaluation and demonstrate our results. Chapter 8 is the conclusion and ideas for future work.

CHAPTER 2

BACKGROUND

"SLAM is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it." D. G. Lowe 2004 [2]

A device may have different sensors (camera, inertial measurement unit(IMU), laser) which perceive the environment. A device needs to know the position and the rotation relative to the map in order to extend the map, so the device has to keep relocalizing in an incomplete map. Problem is that sensors are not accurate enough and have noise. SLAM can be divided into different categories based on precision, types of sensors, the map it generates and many others.

Pose

Pose store information about translation and rotation. It represents transformation between object local coordinate and world coordinate. Usually, it is represented by a 4x4 matrix.

Anchor

The anchor is a pose which we use to anchor another object relatively against. It is meant to be shared in time and between devices and systems.

Sensors

We introduce two basic sensors here.

GPS(Global Positioning System) is one of the most basic sensors for localization. It can give us geolocation, but precision is in few meters. Another problem is that the GPS signal is not stable. In a shopping mall or underground parking the signal can be lost completely. In cities, reflections from the surrounding buildings can interfere with the signal. Moreover, GPS only gives us a translation, so we are still missing a rotation.

IMU is an acronym to an inertial measurement unit. It is used to compute the rotation and translation of the device. It usually consists of accelerometers, gyroscopes, and magnetometers. An output from IMU is frequent, usually tens of Hertzs and data have small errors. Problem is when we use it for a longer time then an error accumulates. IMU results have offset changing slowly over time. We get rotation by integrating angular speed from gyroscope and translation by integrating acceleration from an accelerometer. In this process error sums up.

We can run SLAM on a mobile device, but also on the server. Both of these approaches have different trade-offs and we would like to combine them to create a more accurate and more robust system.

Loop Closure

As we observe the environment our estimation of position drifts apart as error sums up, this can cause that we have done a loop and in reality, we ended up on the same position, but our position estimate is shifted. As we enter an area with known features, we can use these features to correct position estimate.

SLAM on the mobile phone

On the device, we have IMU so we can use this sensor and have instant information about changes of pose and in combination with building map from images, we can have an instant change of pose with minimal error. Problem is when this small error accumulates and we drift apart, we can use loop closures to suppress this problem, but it's limited since limited memory and processor time. On the device the map is only local, so we can localize only this device and other devices can't see the object we added by this device. So we basically have to build some global map or do something else to localize our device with other devices and objects in our space.

We introduce a few libraries we use in our system.

ARCORE[1]

Arcore is implementation on SLAM on the mobile phone for Android, Google released the first version of this project in March 2018.

This library helps with:

- motion tracking: tracking of relative position in the world
- environmental understanding: detection of planes and surfaces,
- light estimation

CloudAnchors (Arcore)

Part of Arcore. It makes possible to get transformation between multiple maps from different devices which are in the same environment. It has a simple interface to implement, but it gives you not much control over system and possibilities are limited.

ARKIT (similar to Arcore)

SLAM for iOS from Apple. So it is possible to port our android app to iOS.

SfM on the server

On the server, we can build one persistent global map for all devices. We have data from all devices on the server, so we have more data and more computation power, so we can do more optimization and have better accuracy. Also when estimating position in time, we already know images from earlier. When we want to localizes device in this map, we send request from a device with an image which camera sees, server localization this image and returns pose of the device, it all can take few seconds regards to the connection. The global map makes possible to store the position of shared objects and retrieve them to the device. There are multiple open-source implementations already as OpenSfM, OpenMVG.

Feature maps

Map generated by SLAM can be represented as poses of features or landmarks in the environment. Feature maps are good for the environment with recognizable landmarks. There are basically three types of features depends on the sensors we use.

Feature types:

Range only - It can be obtained by the strength of the wifi signal.

Bearing only - It usually obtains by a monocular camera, so we have information about direction, but we don't have any information about depth.

Range and bearing - It can be obtained by the stereo camera, these features have established position relative to our device.

In our project, we work with a monocular smartphone camera, so our features are visual landmarks. Information about positions of these landmarks can be obtained by triangulation, that means that we don't know much after we first time see this feature. We can compute its position after we see the same landmark from different angles. To extract features from camera images there are multiple feature extractor. One of the most known is SIFT [3].

CHAPTER 3

END-TO-END SOLUTION

A general overview of the solution

To be able to share object positions between devices, we need some global coordinate system, where we can store positions of these objects. To achieve this goal we build a global map of the environment on the server. On the phone, we build a local map so we know relative change of our position. The problem is to find a transformation between the server map and the client map. To obtain this transformation client sends a request to the server. This request contains an image from the camera. The server runs localization on this image and it finds the position of the phone when the image was taken. From the moment the client has this transformation it can work with global coordinates. The problem is that server response has latency and frequency of response is around every few seconds, so this information is not real-time and it is delayed. To resolve this problem we use SLAM on the client to find how position changed since the last server response. When the client wants to add an object it computes global position from transformation and from the position in local space and it sends a request to the server, the server takes request and saves it to the map. Another part of our system is the 3D web visualizer. Visualizer is a tool to visualize point cloud of the map, user added objects, positions of phones and paths of phones. The server streams information to visualizer and all clients about positions of objects and about changes in these positions. A visualizer is an important tool for debugging and also for further analysis of our system.

Underlying core technologies

The server has two main purposes to build a map and localize clients on this map. On the server side, we decided to use OpenMVG. OpenMVG is open Multiple View Geometry library. It's the library for 3D computer vision and Structure from Motion. We choose this library because it's

open source and it implements a pipeline to compute structure from motion(SfM) and localization.

On the client, we need to compute online SLAM. We use ARCore for this purpose. Arcore is a library for Android from Google. It is supported on most of the new Android phones. Arcore uses phone sensors and images from the camera to build SLAM to provide pose in local map with low latency. Arcore also gives us calibration of the phone camera which we need to be able to do more precise computation.

Visualizer needs to render 3D data and work with 3D data, transformations and matrices. Visualizer runs in the browser, so we chose Three.js for these purposes. Three.js is a cross-browser javascript library which uses WebGL to render 3D computer graphics.

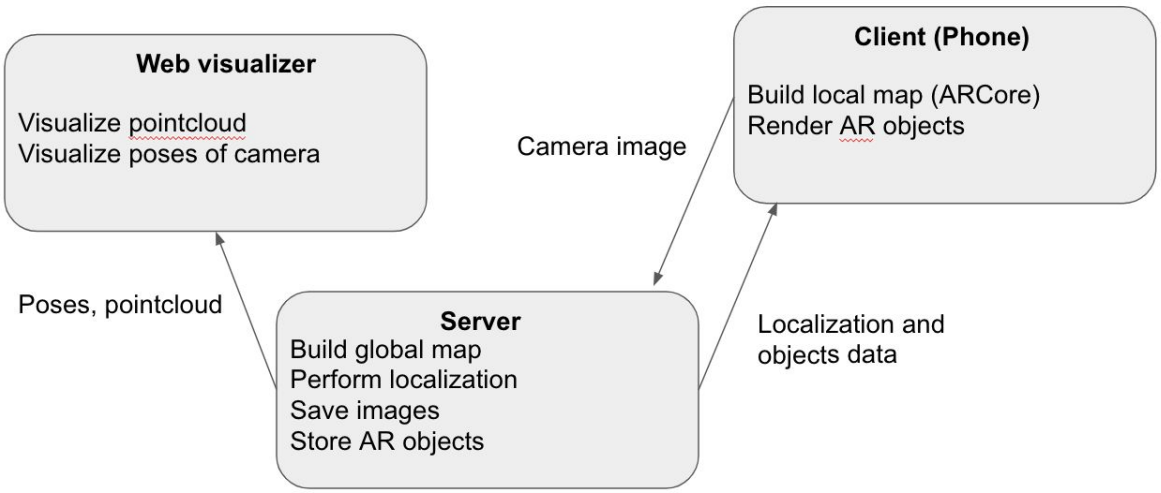

Fig 6. High level architecture of the system

CHAPTER 4

SHARING OF POSES AND CONTENT

In this Chapter, we discuss the matrix representation of pose and transformations. Then we discuss the way how to transform poses between global and local map. We work with poses in 3D homogeneous coordinates.

Translation

The translation is transformation which moves every point by some fixed vector **v**. Translation by vector **v**, can be represented as matrix T_v :

$$T_v = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To use this transformation we can multiply point with this matrix:

$$T_v p = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix}$$

The inverse can be obtained by the reverse direction of vector **v**.

$T_v^{-1} = T_{-v}$

Scaling

Scaling is a transformation which enlarges an object by a scale factor. The scale factor can be different for every axis.

Matrix representation of scaling:

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling of point **p** can be obtained by matrix multiplication:

$$Sp = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x * x \\ p_y * y \\ p_z * z \\ 1 \end{bmatrix}$$

Rotation

Rotation can be represented in the format:

$$R_{4x4} = \begin{bmatrix} & R & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rotation matrix is an orthogonal matrix, so $R^T = R^{-1}$. It follows that the inverse of this matrix can be easily computed.

Pose representation

The pose can be described by rotation and translation. To represent a pose we apply rotation and then the translation to the object. As matrix multiplication is associative we can compose translation matrix and rotation matrix to one matrix which represents pose and can be easily constructed by translation and rotation matrix.

$$TR = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & R_{3x3} & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & R_{3x3} & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation between server and client map

We have pose L in the local map which we get from SLAM on the client. We also have pose G in the global map which we get from server localization. To be able to transform local

coordinates to global coordinates and another way around, we need to know the transformation between these two spaces. Let's call this transformation M. M says how to transform pose from local coordinates to global coordinates which says this equation $LM = G$. That means we can compute M as $M = L^{-1}G$. When the user wants to add object on someplace in the local map, we transform local pose (L_{pose}) to global pose (G_{pose}) $G_{pose} = L_{pose}M$ and we send G_{pose} to server and server save this object to the global map. When the server sends out positions of objects in the global map to the client, the client wants to compute the local pose of these objects and then visualize them. The client computes L_{pose} as $L_{pose} = G_{pose}M^{-1}$.

Content Sharing

When the user wants to share the object with another user it is really simple since at the moment of creation we know not only local object position but also a global object position which is sent to the server and stored in the database. At the time of retrieval, this global position is sent to the other user who now knows the global pose of object and camera as well, therefore can render it at the right place on the screen.

CHAPTER 5

BUILDING MAP

In this chapter, we describe the process of building a map on the server.

Image Listing

At the start, we need to collect images which will be used to build a map. We use the phone which stream images to server and server saves them. To be able to perform SfM we need intrinsic parameters of the camera which took images. If our sensor is in our sensor database, we can obtain intrinsic parameters from there, otherwise, we have to obtain them on our own. In our case, we use ARCore calibration to get these data.

Feature Extracting

In this part, we extract features from every image separately. For feature detection, we use SIFT[3]. SIFT features are invariant to scale and rotation. SIFT try to detect and extract interesting points in the image and describe them. The problem is when the scene is homogeneous and SIFT detects no interesting points. On figure 1, we can see there are 352 detected features, especially in the border of objects. On figure 2, we can see that the wall is homogeneous and there are not found any features and on the image, there are 62 detected features and there is a white wall which doesn't contain any of them. These features are used for

Fig 1. Detected features around bed

Fig 2. Detected features on the wall

matching. A low number of matched features makes the approximation of camera pose less precise.

Feature Matching

Feature matching is the process of finding visual overlap on images. From the feature extracting phase we have features of images. The feature is described by features descriptor. The descriptor is usually an array of floats. In feature matching phase we would like to take two images and match features which describe the same object just from another perspective. To be able to say which features are similar we need to use metric, most used metrics are hamming and Euclidean. When we have feature descriptors as $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$, the hamming metric is defined as $\sum_{i=1}^{n} \mid p_i - q_i \mid$ and the Euclidean metric as $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$

To determine k-nearest neighbors of the feature, we can use brute force and compute distances between every pair, this method is time ineffective. To determine k-nearest neighbor we use FLANN(Fast Library for Approximate Nearest Neighbors)[8]. Our algorithm takes feature of the first image(let's call it X) and try to find 2 nearest features (let's call them - Y(nearest), Z), in the second image, then it takes the Y and finds the nearest feature of the Y in first image if this feature is the same as X then we suppose we found a match. It also checks the ratio of distances XY and XZ. We consider match valid only if this ratio is bigger than the threshold. We repeat this process for certain pairs of images. Our source of images is basically a video that means we have images in order and images which are close to each other are more promising to find matches, so we try to find matches only between these images. After this process, we have putative matching.

Next step is geometric filtering. Geometric filtering is used to preserve geometric consistency. We want to somehow estimate camera poses of these two images and then project matched features to create 3D points and check if we can create these 3D points correctly. For this process, we use Ransac[9] algorithm. Ransac picks a few samples of matched features and estimate camera poses, then project other matched features and check if the 3D point was created

correctly, so it counts inliers. Ransac repeats this process with other random matched features. After some number of iteration, it chooses a pair of camera poses which have most inliers. At the end of feature matching, for some pairs of images, we have a relative transformation between camera poses of these images and information about which features belong to which feature in the second image and 3D points. On figure 3 we can see two images and feature points which were matched

Fig 3. Matched features

SfM Algorithm

This is the main process of building a map. From the feature matching, we know mutually camera poses of images. Our SfM algorithm is iterative. At the start, we take one pair of camera poses. It is important to pick the right one as we would like to have a stable anchor. We can pick based on the angle between poses, the number of inliers, the error of inliers. Then we estimate 3D points as a projection of features and add another camera pose from another image, we know exact rotation from feature matching, but translation is up to scale, so we estimate position based on 3D points as we know which feature belong to which 3D point. We can measure error as the distance of 3D points from projected features. Our total error is the sum of squares of errors. We would like to minimize the total error. We use bundle adjustment. Bundle adjustment[11] is a process of optimization. It takes poses of cameras and 3D points and adjusts it to minimize the total error. One well-known minimizer is Ceres Solver[10]. We repeat this process until we add all camera poses. After this process, we have estimated camera poses and a 3D point cloud of the environment.

The problem is scaling as the algorithm cannot estimate the scale of pointcloud since it does not know the size of objects. After algorithm build map, we are obliged to fix the scale. So we find in the map an object with known size in the real world and measure it in the point cloud. Then we compute the scale factor between real-world size and map size. When we know the scale factor, we apply a scale factor to transformation.

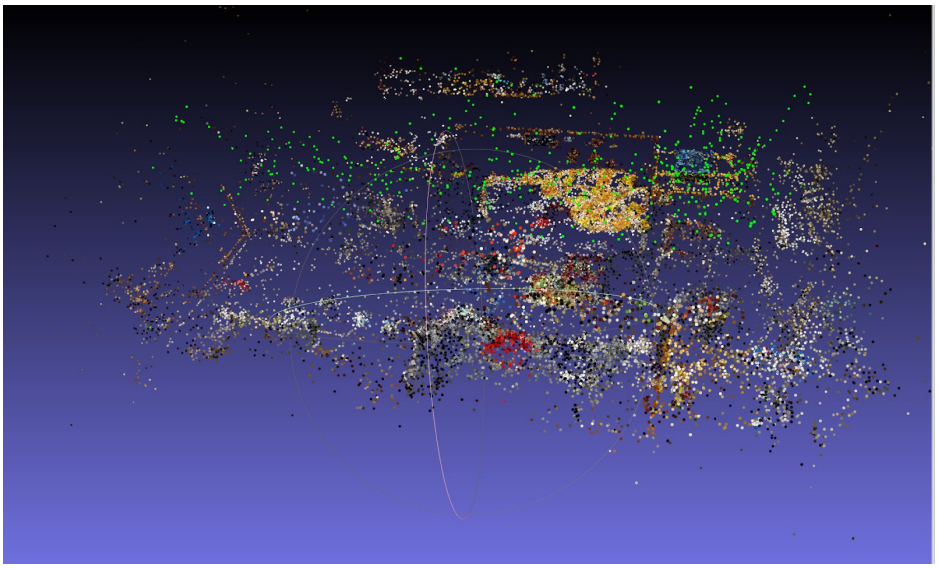

Fig 3. Generated point cloud

CHAPTER 6

SOFTWARE DESIGN AND IMPLEMENTATION

In this chapter, we discuss the implementation details of our system.

Server

On the server, we need to create simple HTTP server which supports WebSockets and async programming. We need async programming because we call localization process from inside and it can take a few seconds to get output so it would be useful if in the meantime server can serve other requests. We decided to use WebSockets protocol for communication because we would like to stream data from client to server but also another way around. Our decision was to use Javascript and NodeJS. For WebSockets communication we use library socket.io, it's useful because it makes work with WebSockets much more comfortable. We send all data in JSON format. JSON is easy to use, easy to debug and it's supported on Javascript and Android. The main purpose of the server is to provide localization.

Endpoints:

- Cam_img_phone:

  It takes care of providing localization. This endpoint takes data which contains an image from phone camera and metadata as request id, hashCode of device and time when the image was taken. At first, it saves the image and runs the separate process with openMVG localization binary and wait for results. After localization is done, our server parses JSON output. JSON output pose of the image in the global world, after it is parsed server adjusts data to respond and sends data to the client. The server saves every image on the disk, these data can be also used when we want to build a new map, that means we can use this endpoint to create data for building map purposes.

- Add_android:

  It takes the pose of the new object and saves it to memory and stream these data to visualizer and other phones.

- Cam_pointcloud_phone:

  It takes an array of new point cloud points from the client and streams them to the visualizer.

- Cam_pos_phone:

  It takes the position of the client and streams it to the visualizer.

Client

The client is an android phone. We use ARCore for computing SLAM and application is coded in Java. After every frame ARCore computes new position, we send this position to the server for web visualizer so we can see moving client on the visualizer in real time. Every 4 seconds we also send a camera image for localization to obtain new transformation between server and a local map. The reason of 4 seconds delay is because localization usually takes a few seconds and also image has around 1 MB, so it takes time to deliver these data to the server. When the response of localization is received, the server computes transformation between global and local map and adjust positions of objects. ARCore can detect planes, when the user clicks on planes in our application it computes position on that plane in the local map, then it uses transformation to transform these pose to global map and send the request to the server to save this object in the global map. On figure 4 we can see our app screen. On the view, we see two user objects and white dots visualize detected plane.

Web visualizer

The main purpose of the visualizer is real-time visualization of positions and phones and also show global map and added objects in it. We developed visualizer in javascript. Javascript is the main language in web development. For the component structure of the application, we used a well-known framework React. Visualizer takes data from the server via WebSockets. Visualizer consist of multiple smaller react component, every of this component has only one purpose. The main scene shows the most important data, it shows all connected phone and user objects. To show the path of the device, it draws a line between every two consecutive positions so we can track the whole ride of device. It also shows the frustum of the camera so we can see the rotation of the camera and what camera can see and what cannot. On the right up corner, we can see Axes helper. It helps us see how is the main scene rotated. On the right down corner, we see the last image from the device. On the left up corner, there are few options, we can set up if we want to

see point cloud, features of image and axes helper. It makes the whole experience of visualizer more pleasant. On figure 5 we can see the screen of web visualizer.

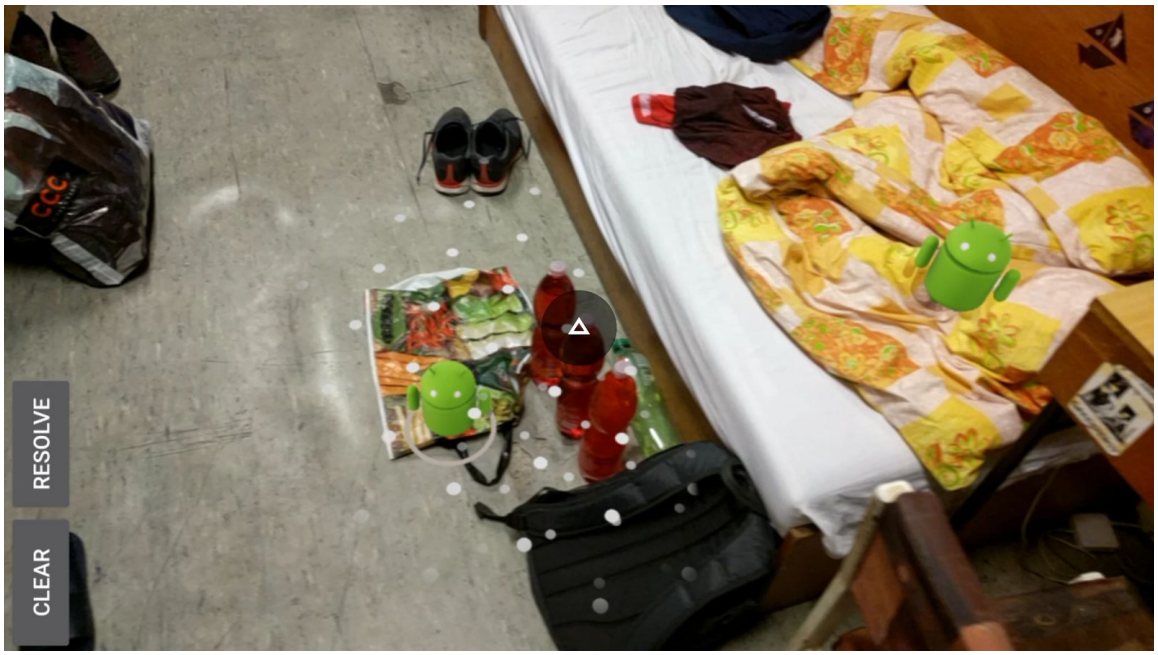

Fig 4. Client app view

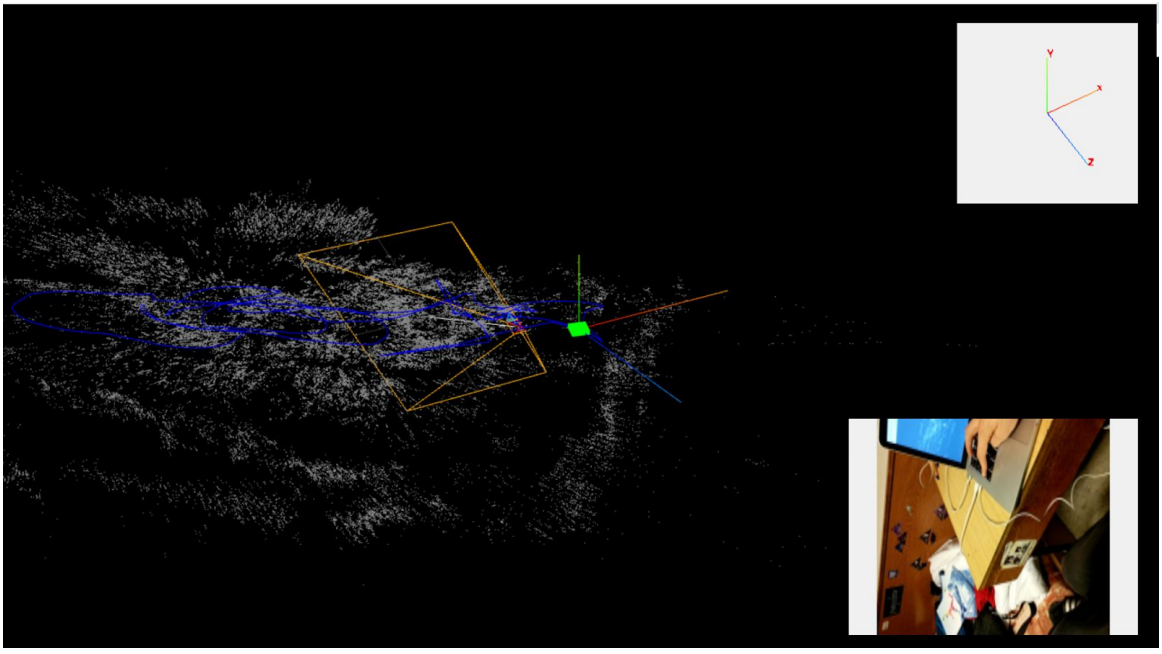

Fig 5. Web visualizer

CHAPTER 7

EVALUATION AND METRICS

In this chapter, we discuss the evaluation, metrics and our results.

Metrics

To evaluate the accuracy of localization we can measure multiple metrics. Precision and recall are most used. In our case precision says what percentage of our predictions were correct. In our case recall says what percentage of requests we were able to respond. We would like to also measure the latency of localization, it says how long it takes to send an image from client to server and compute localization and send a response back. We can also measure the build time of the global map and parameters of the global map as a number of points in the point cloud. When we want to measure the accuracy and quality of the map we could use other technique(like 3D scanner or Kinect) and then compare results.

Results

As we stated at the assignment, we created an end-to-end solution for multiplayer shared augmented reality. Users are able to add AR objects to map and other users can fetch them and see them in the same position. There is a common problem in other feature-based approaches, like in ARCore that every new anchor incurs CPU cost[12], whilst our system can embrace virtually unlimited number of anchors since we have a global map and for every new object we only remember one pose matrix on the server and those matrices can be served to the client in chunks based on the client position.

Our system consists of a server which is able to build offline SfM map and localize images from the client app. We also have a client mobile app which is able to communicate with the server to have more accurate localization and to get positions of AR objects which can render to view. We test our solution in the room. Size of this room is 3m x 5m. For building map, we used Amazon EC2 instance t3.large. This instance consists of 2 cores processor and 8GB of

RAM. We built our map from 438 images. Building the map took 33 minutes. Poses of 345 images were estimated. We put the mobile in the different poses and sent a request to localization. We tried to localize 68 images. Localization managed to find pose in 62 requests, so recall is 91%. Median of latency is 3.44 seconds. Our 3D point cloud consists of 30268 points. On figure 7, we can see the histogram of latency of localization in seconds.

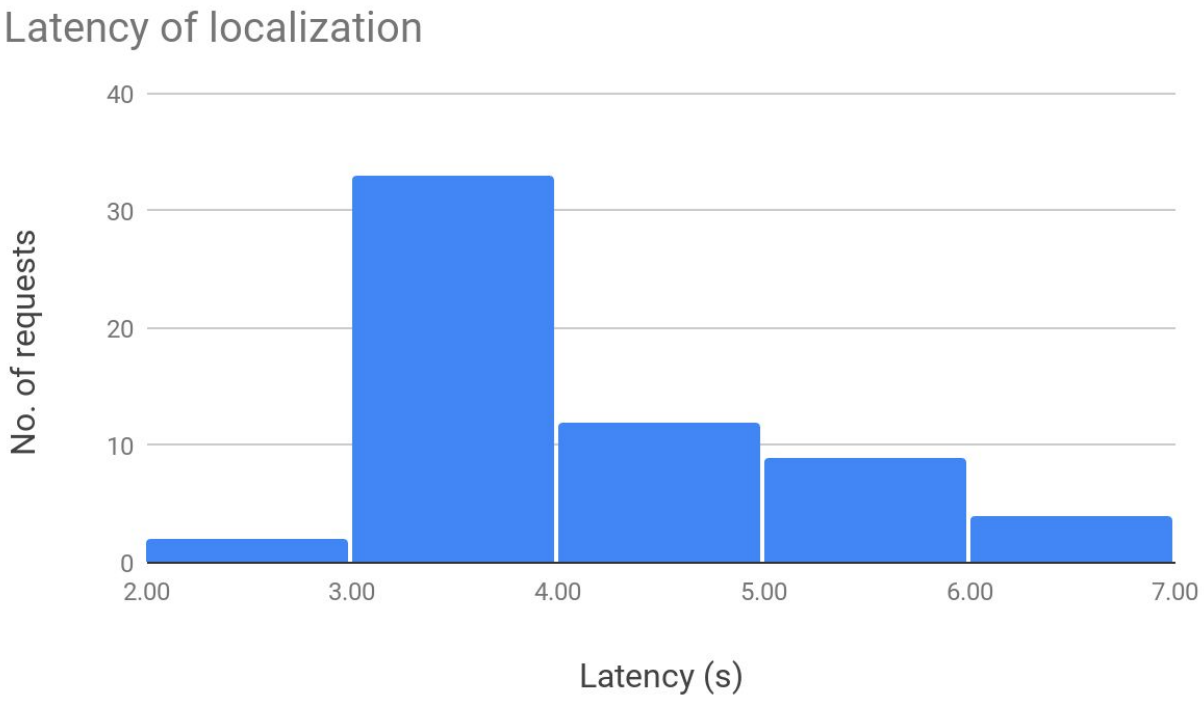

Fig 7. Latency of localization

CHAPTER 8

CONCLUSION

We accomplished to build a basic system for sharing objects and positions of devices in an unknown indoor environment. Building map and SLAM is stable and we were able to build good enough map from just 438 images in 33 minutes.

After every server localization, we can see a visible jump in position, that means that SLAM on client drifted apart from the global map from which we infer that low latency of server localization and smoothing of current location estimates is critical for the accuracy of our system.

Future work

The latency of localization:

The latency of localization is a critical problem. It usually takes a few seconds to send an image to server and localize it in the global map. Localization could be accurate but after few seconds phone moves a lot and it can be inaccurate again. In the current solution for every localization, we run the whole new process with localization binary. This binary at first loads features and data and then it provides localization. It could be useful to run it and load features and data only once and then start HTTP server which can do only localization itself. Another improvement would be to use a more compact format than JSON so serialization can be more efficient. We could use gRPC for high-performance communication and protobuf as a protocol for serialization.

Automatically scale estimation:

The problem is that after we build the map we have to estimate scale manually. To automatize scale estimation, we can use ARCore. Our SLAM on the server does not know anything about scale but SLAM on mobile phone uses IMU sensors, which can be used to estimate scale.

Smoothing of current location estimates:

In the current solution when we get a new response we compute the transformation and move every object to a new position which we computed using new transformation. This has an effect that after every response our server shifts a little bit. We would rather have some continuous transition. So we would like to implement weighted position and rotation averaging to make the transition smoother. We also have to deal with the possibility that localization provided by the server can be completely off since in the current solution we do not filter them out.

Literature

[1] Google. Arcore overview. https://developers.google.com/ar/discover/, 2018.

[2] Wikipedia. Simultaneous localization and mapping — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping, 2019.

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91–110, November 2004

[4] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. pages 1403–1410, 2003.

[5] Ikea. Ikea place app. https://highlights.ikea.com/2017/ikea-place/, 2018.

[6] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. CoRR, abs/1606.05830, 2016.

[7] Bgame. AR mole app. https://play.google.com/store/apps/details?id=com.BGame.MoleCatchAR&hl=en_US, 2018.

[8] Muja, Marius, and David G. Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration." VISAPP (1) 2.331-340 (2009): 2.

[9] Yaniv, Ziv. "Random sample consensus (RANSAC) algorithm, a generic implementation." Insight Journal (2010).

[10] Agarwal, Sameer, and Keir Mierle. "Ceres solver." (2012).

[11] Triggs, Bill, et al. "Bundle adjustment—a modern synthesis." International workshop on vision algorithms. Springer, Berlin, Heidelberg, 1999.

[12] Google. Arcore anchors. https://developers.google.com/ar/develop/developer-guides/anchors/, 2018.