Aalto University
School of Science
Degree Programme in Security and Mobile Computing

Marius Noreikis

Image Based Indoor Navigation

Master's Thesis
Espoo, June 30, 2014

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Professor Peter Sjödin, KTH Royal Institute of Technology
Advisor: Dr. Yu Xiao

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF
MASTER'S THESIS

Author:	Marius Noreikis		
Title:	Image Based Indoor Navigation		
Date:	June 30, 2014	**Pages:**	87
Major:	Data Communication Software	**Code:**	T-110
Supervisors:	Professor Antti Ylä-Jääski Professor Peter Sjödin		
Advisor:	Dr. Yu Xiao		

Over the last years researchers proposed numerous indoor localisation and navigation systems. However, solutions that use WiFi or Radio Frequency Identification require infrastructure to be deployed in the navigation area and infrastructure-less techniques, e.g. the ones based on mobile cell ID or dead reckoning suffer from large accuracy errors. In this Thesis, we present a novel approach of infrastructure-less indoor navigation system based on computer vision Structure from Motion techniques. We implemented a prototype localisation and navigation system which can build a navigation map using area photos as input and accurately locate a user in the map. In our client-server architecture based system, a client is a mobile application, which allows a user to locate her or his position by simply taking a photo. The server handles map creation, localisation queries and pathfinding. After the implementation, we evaluated the localisation accuracy and latency of the system by benchmarking navigation queries and the model creation algorithm. The system is capable of successfully navigating in Aalto University computer science department library. We were able to achieve an average error of 0.26 metres for successfully localised photos. In the Thesis, we also present challenges that we solved to adapt computer vision techniques for localisation purposes. Finally we observe the possible future work topics to adapt the system to a wide use.

Keywords:	Navigation, structure from motion, point clouds, feature extraction, indoor localisation, navigation mesh, pathfinding
Language:	English

Aalto-universitetet
Högskolan för teknikvetenskaper

Aalto-universitetet
Högskolan för teknikvetenskaper
Examen i Säkerhet och mobildatakunskap

SAMMANDRAG AV
DIPLOMARBETET

Utfört av:	Marius Noreikis		
Arbetets namn: Image-baserad inomhusnavigation			
Datum:	Den 30 Juni 2014	**Sidantal:**	87
Huvudämne:	Datakommunikationsprogram	**Kod:**	T-110
Övervakare:	Professor Antti Ylä-Jääski Professor Peter Sjödin		
Handledare:	Dr. Yu Xiao		

Forskare har de senaste åren framfört olika inomhusnavigations- och lokaliseringssystem. Dock kräver lösningar som använder WiFi eller radiofrekvens identifikation en utbyggdnad av stödjande infrastruktur i navigationsområdena. Även teknikerna som används lider av precisionsfel. I det här examensarbetet redovisar vi en ny taktik för inomhusnavigation som använder sig av datorvisualiserings Structure from Motion-tekniker. Vi implementerade en navigationssystemsprototyp som använder bilder för att bygga en navigationskarta och kartlägga användarens position. I vårt klient-server baserat system är en klient en mobilapplikation som tillåter användaren att hitta sin position genom att ta en bild. På server-sidan hanteras kartor, lokaliseringsförfrågor och mättningar av förfrågorna och algoritmerna som används. Systemet har lyckats navigera genom Aalto Universitets datorvetenskapsbiblioteket. Vi lyckades uppnå en felmarginal på 0.26 meter för lyckade lokaliseringsbilder. I arbetet redovisar vi utmaningar som vi har löst för att anpassa datorvisualiseringstekniker för lokalisering. Vi har även diskuterat potentialla framtida implementationer för att utvidga systemet.

Nyckelord:	Navigation, Structure from Motion, punktmoln, egenskapsextraktion, inomhusnavigation, navigationsnätverk, vägsökning
Språk:	Engelska

Acknowledgements

First of all, I would like to thank my advisor Yu Xiao for guiding me through the whole process of writing my Thesis. Her insightful comments and observations helped me to solve many of the problems encountered.

I would also like to express my sincere gratitude for giving me a chance to work on this Thesis topic. I would like to thank Computer Science and Engineering department and my supervisors Prof. Antti Ylä-Jääski and Prof. Peter Sjödin. I thank NordSecMob Consortium for this wonderful opportunity to study in the programme which gave me invaluable experience and helped to achieve important goals.

Finally, I want to thank my family for all the support during my studies.

Espoo, June 30, 2014

Marius Noreikis

Abbreviations and Acronyms

3G	Third generation of mobile telecommunications technology
EXIF	Metadata information associated with an image
JPEG	Raster image format defined by Joint Photographic Experts Group
JSON	JavaScript Object Notion
LTE	Long Term Evolution; Wireless broadband technology
NFC	Near Field Communication
REST	Representational State Transfer; Popular web service software architecture
RFID	Radio Frequency Identification
PNG	Portable network graphics; raster image format
SIFT	Scale-invariant Feature Transform
SfM	Structure from Motion

Contents

Chapter 1

Introduction

When the Global Positioning System (GPS) was made available to public, it created a way to develop numerous new applications that we now use in our everyday life. Navigation systems are amongst the most important ones. Consumer and industrial vehicles, planes and smartphones integrate GPS navigation systems that help to guide a user. Due to high availability all around the globe and good accuracy, GPS systems almost completely changed the way we navigate to our travel destinations.

However, GPS have certain restrictions. While being highly accurate to detect a position of a GPS receiver outdoors, the system is unable to work in shielded environments such as indoors, caves or tunnels. This is due to the receiver which requires a clear line of sight to the GPS satellites to work.

During the last few years, various methods were proposed for indoor positioning systems, such as usage of the mobile cell ID, WiFi or Bluetooth signal strength, or dead reckoning techniques. N. Fallah et al. identifies four different types of localisation techniques in current indoors navigation systems: triangulation, pattern matching, direct sensing and dead reckoning [13]. All the four techniques provide a way to detect a current user position and to track the user in the area, though none of the techniques is universal and significantly better than others.

Triangulation technique is similar to the one used with GPS but regarding closed environments, other types of wireless signals are used, such as WiFi signal strength, cell ID from the mobile cell towers and base stations or Radio Frequency Identification (RFID). Cell tower positioning is often used with GPS enabled systems to supplement the system in areas where the GPS signal is too low [3]. However, even with the advanced techniques to improve the accuracy, the error of cell ID based systems for locating the user can be up to several hundred meters [3]. WiFi and other wireless radio based solutions can provide very accurate results but require additional infrastructure.

An example of direct sensing is an RFID based system. Even though it provides high accuracy and pose estimation [25], direct sensing requires unique tags scattered all over the navigable area, often in a controlled way. Each tag uniquely defines the position of the user as soon as the tag is reached and scanned. Tags can also be based on Ultrasound or Bluetooth technologies. However, the requirement of installing tags in the environment makes the system more expensive and not widely available.

Dead reckoning techniques estimate the location of the user based on previously recorded location using devices such as accelerometers, gyroscopes and electronic compasses. Fisher et al. presented a dead reckoning system which uses foot-mounted inertial sensors and ultrasound sensors to reliably estimate the position of the user [14]. However, such an approach quickly accumulates errors and is still dependent on synchronisation beacons.

Pattern matching techniques rely on computer vision algorithms or pre-recorded signal distribution and fingerprinting maps [13]. Computer vision algorithms include image matching to a pre-recorded database of images or to a pre-generated 3D model of the area. Usually, solely computer vision based systems require no additional infrastructure deployed in the navigation area. That is, the systems require no RFID or WiFi fingerprinting, neither utilisation of any other radios nor tags. Such systems can be deployed in most of the indoor areas without installing any additional hardware in that area. A system that is able to work without the mentioned additional infrastructure has several important advantages. Firstly, there are no incurred costs of deploying and maintaining the infrastructure. Secondly, infrastructure-less system does not suffer from downtime or degraded accuracy due to infrastructure hardware failures. Additionally, an image based indoor navigation system manipulates information obtained using image sensors, therefore it can provide visual feedback to the users. For human beings, it is easier to understand navigation instructions that show the real world views than to read 2-dimensional (2D) floor plans or maps.

In a recently carried Indoor Localization Competition by Microsoft[1], 22 teams of researchers presented their state-of-art solutions for indoor positioning. Most of them were based on WiFi fingerprinting but other solutions included beacons that use 2.4GHz radios, ultrasound or even Light Emitting Diode (LED) light. Other approaches exploited combined techniques where they used WiFi fingerprinting along with smartphone sensors to obtain an accurate position. Most of the participating teams were capable of achieving less than 5 meters average location error. Bestmann et al. demonstrated a very high accuracy with less than 1 meter localisation error [6]. However, all

[1]http://research.microsoft.com/en-us/events/ipsn2014indoorlocalizatinocompetition/

the solutions rely on additional infrastructure, requiring at least WiFi access points to be deployed in the area.

To the best of our knowledge, currently there is no image based indoor navigation system that would rely only on computer vision based techniques to build a navigation map and localise positions of users. Thus, the Thesis work mostly focuses on developing a prototype image based navigation system. The developed solution is fully functional and implements all the necessary functionality to provide indoor navigation service for mobile devices. The effort is made not only on the implementation of a working system but also on testing and evaluating the solution in a real world environment.

1.1 Problem statement

There is no reliable way to navigate users in the indoor environments without utilising extra infrastructure. Our goal is to create an indoor navigation system that would locate a current position of a user within an accuracy of several meters, would not require infrastructure and would work with widely available consumer devices, such as smartphones or wearable gadgets. The input for the system should be photos taken by smart devices. The system would generate a navigation map and would be able to localise a position of a user in the map. The system should also plan a path between the current position of the user and a selected destination and give visual instructions for the user to help him or her to navigate through the area.

1.2 Methodology

We used an experimental approach to implement a prototype of an image based indoor navigation system. Before the implementation, we developed a set of functional and non-functional requirements for our new system. After surveying existing indoor navigation systems, we selected the main technologies to be used for the system implementation. The prototype system combines techniques of computer vision methods as well as navigation algorithms used in virtual reality projects (e.g. contemporary computer games).

A computer vision technique to build a 3-dimensional (3D) model from photos is called Structure from Motion (SfM). The model is presented as a set of 3D points that correspond to the shape of an object seen in the photos. Structure from Motion methods were chosen due to their unique ability to accurately recover a position and orientation of a camera that takes a photos in a known area. Other supplementary algorithms to adapt

SfM for navigation purposes were chosen according to their functionality and compatibility with the existing code base. More detailed descriptions of why a particular component was chosen are presented in chapters 4 and 5.

After creating the system, we run several tests to evaluate the performance of the system and to confirm that it meets the functional and non-functional requirements. Firstly, we evaluated the accuracy of the positioning algorithm. Secondly, we evaluated the time needed to build navigable maps of areas and to process location queries.

The experimental approach is suitable for such kind of research due to several reasons. During the development of prototype, new challenges and issues were raised and solved, which may not be easily visible while designing a system only from a theoretical perspective. The result of the work can be easily evaluated not only in laboratory conditions but also in real world scenarios.

1.3 Structure of the Thesis

The second chapter of the Thesis explains a brief background and current researches in the field. The Background section gives an overview of computer vision algorithms and navigation algorithms, as well as the main concepts used in the Thesis. The Related work section briefly sums up some of the notable researches in the fields of computer vision based location recognition and indoors navigation.

Chapter 3 describes the main parts of the system and a real world use case where our navigation system would be applicable. Chapters 4 – 5 focus on system implementation details and challenges. The system can be logically divided into two parts: the first part is about creating a navigation map and finding user's position using computer vision techniques and the second is about navigation and path finding. Chapter 4 presents the solutions to the challenges of SfM model creation, model growing and localisation. Chapter 5 describes the issues that were solved to create a suitable navigation map, find paths in the created map and generate navigation instructions.

Chapter 7 presents an evaluation of the created software system. Chapter 8 discusses the challenges and lessons learnt during the development of the navigation system and possible ways to improve it in the future. The thesis ends with conclusions and a list of cited literature.

Chapter 2

Background and related work

This chapter introduces the work done by other researchers in the fields of computer vision based localisation. It also gives some basic background about computer navigation algorithms.

Section 2.1 describes background and related work of image based localisation systems. Section 2.2 describes in detail the Structure from Motion model building method. Section 2.3 briefly explains navigation algorithms and the ways of building a navigation map.

2.1 Image based localisation

Researches focusing on image based localisation extensively use computer vision techniques of pattern matching and building 3D models. Before describing the work done in the field, we will give an introduction to a terminology related to computer vision techniques that are used to enable image based localisation.

2.1.1 Main terms

This section gives a basic background about the computer vision terms and techniques used in the Thesis work. It describes concepts of image features and point clouds.

2.1.1.1 Features

A feature can be thought of as an interesting point of the object and a set of features can fully describe a particular object. For the 2D images, features are places of the image which have a high gradient change in neighbouring pixels. Features can be found near the edges of objects inside the image or in

places where the image has a rich texture. If an image is of one same colour, no features can be found. In different literature features can also be referred to as keypoints. Image features are a basis for computer vision algorithms for building point clouds.

2.1.1.2 Point clouds

When talking about 3D models that are reconstructed using computer vision techniques, researchers usually refer to the model calling it a point cloud. A point cloud is a set of 3D points which represent points of a 3D object. Typically, each point of the point cloud has 3 (x, y, z) coordinates. It can also contain information such as the colour of the point, the estimated error of the point position or the number of measurements that were used to estimate the position of the point. For example, a point in an *.NVM* point cloud format has the following information: X, Y, Z coordinates expressed as floating point numbers, RGB colour value expressed as 3 integers and a list of feature locations that were used to estimate the position and colour of the point.

Point clouds can be classified as sparse and dense clouds. An example of a sparse and a dense point cloud is presented in Figure 2.1. Dense point clouds have substantially more points than the sparse ones. As seen in the Figure 2.1, if the cloud is dense enough and its points contain colour information, it is easy to infer the shape and recognise an object by simply looking at such point cloud.

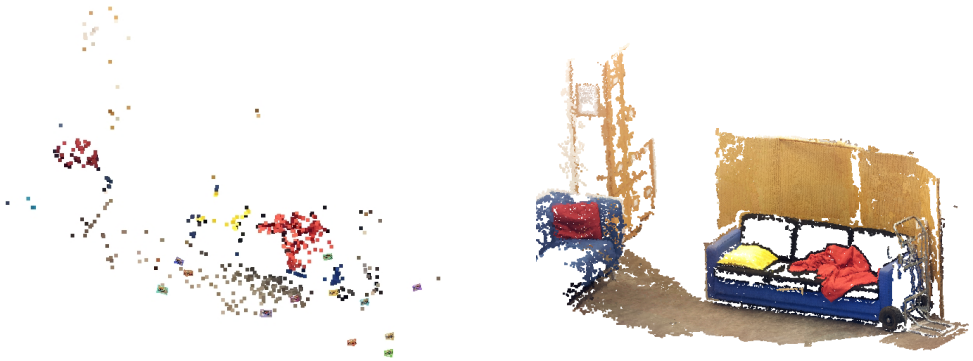

Figure 2.1: On the left: objects represented by a sparse point cloud; on the right: the same scene represented as a dense point cloud

2.1.2 Related work

This section briefly describes the most notable work done on computer vision based localisation. The first research described in this section was an early attempt to use only computer vision pattern matching methods to localise positions. The other researches use more advanced techniques and preset truly infrastructure-less approaches of localising a position of a user.

Image matching to a pre-recorded database of images was used by Kim et al. in their computer vision based location positioning and augmented reality indoors navigation system [19]. Researchers used sequences of images of the navigable indoors environment taken by a head mounted wireless camera. After building the image database they were able to locate a user with 89% success rate. However, they did not rebuild a 3D model of the area but instead, in order to increase the accuracy of the system used paper printed landmarks. When a mark is detected in a picture taken by a user, the system is able to estimate the position of the user by checking the local database of existing marks. Each mark is mapped to a particular area in the map. Although it is cheap to put such marks in the area, it requires additional work and careful planning of where to place such landmarks.

Fast location recognition based on a 3D point cloud is presented in a research by Irschara et al. [16]. A sparse point cloud is built using photos from well known urban areas. Such photos can be taken e.g. from the Internet photo sharing websites. The research focuses on providing a real time registering of the images to an existing point cloud, thus can even register video frames taken in real time. For location recognition speed-up, the system uses vocabulary tree-based indexing of the features and synthetic views to cover the most parts of the navigable area. The synthetic views help to register images taken from novel angles and help to reduce the point cloud model size. The findings provide a reliable and efficient mechanism of determining the position of the user in the outdoor scenario but can be used in a same way indoors. However, the research only focuses on fast tracking of the images but not reconstructing the navigable area or navigating the user in the area. Nevertheless, a proposed way of storing only useful images and reducing the model size and increasing the speed of locating the user would highly benefit image based navigation systems if the approach could be adapted. Currently, the solution is not open sourced and is not freely available to the public.

Another efficient solution for estimating pose and location was proposed by Li et al. [20]. The algorithm of locating a new photo was tested on models with as many as 800.000 photos. The approach is claimed to be usable for finding locations in a world scale models. The approach is to firstly filter

the results by the visual features that always appear close to each other. Otherwise, single features may appear in many photos that were taken in very different parts of the world. Their matching techniques are based on 2D-to-3D registration method [26] which allows to quickly find corresponding features between an input image and a 3D point cloud. The method uses bidirectional matching, meaning that image features are matched to 3D points and then 3D points are matched to image features. Even though, the algorithm cannot prevent false registrations, the rate of false registrations is very low (0.68% in their experiment). Such technique could allow locating a user in a model that contains many indoor areas. Paired with other simple, though coarse grained navigation techniques such as cell ID triangulation, the algorithm could further decrease the ratio of false registrations and benefit large scale navigation systems.

2.2 Structure from motion

A method of generating point clouds from 2D images is called Structure from Motion (SfM) [30]. The SfM method uses pictures or photos of an object as an input and ideally produces a 3D model of the object represented as a set of 3D points. Building a 3D model out of 2D images requires several steps. Steps include features extraction, features matching, recovering of the parameters of cameras and of positions of the points in a point cloud [29]. The camera in this case is a 2D image and a relative position showing where the device that took the picture was held. The last step of finding the correct positions of the 3D points is also called *Bundle Adjustment* [31]. Below, we present the current and popular methods used for features extraction, matching and bundle adjustment.

2.2.1 Features extraction

Scale Invariant Feature Transform (SIFT) [22] is a popular algorithm used for features extraction. It can extract highly distinctive and invariant features from images [22]. Features are invariant, since the same feature can be found in a picture or photo which is rotated, scaled or of different lighting than the original picture where the initial feature was found. The algorithm can also deal with noise and is not sensitive to affine transformations[1]. It is also suitable for large numbers of features since it is using highly distinctive feature descriptors. Each descriptor is stored in a 128bit vector. The

[1]http://mathworld.wolfram.com/AffineTransformation.html

descriptor size must be large enough, since a relatively small sized image of 500px x 500px size can yield as many as 2000 features [22].

The original SIFT algorithm was used by N. Snavley in his Bundler software [29] but later, various enhancements were proposed to the algorithm for both speed and accuracy improvements. A study by Wu et al [37] compares some of the popular feature extraction algorithms including SIFT [22], PCA-SIFT [18], GSIFT [21], CSIFT [2], SURF [4] and ASIFT [23]. The study shows that even though the original SIFT algorithm works well with changes in scale and rotation, it is outperformed by PCA-SIFT, GSIFT and CSIFT when pictures are blurry or contain a substantial change in an illumination. Although ASIFT algorithm was considered to be the best choice in finding many keypoints under the conditions of large affine transformations, it needs more than 3 times more time to detect the features than other algorithms. SURF was proved to be the fastest algorithm in the group. However, the algorithm was not as good in finding features under different conditions as the other algorithms. Changchang Wu proposed a variant of SIFT algorithm which can run on a Graphics Processing Unit (GPU) [33]. The algorithm called SiftGPU is a parallel and highly optimized SIFT implementation. It can run directly on GPU either using GLSL[2] or CUDA technologies. Even with the GPU support, some tasks of the algorithm are still required to run on CPU. Nonetheless, SiftGPU uses multi-threading and can fully exploit the performance of modern multi-core processors. Codreanu et al also implemented a SIFT algorithm which can run on a GPU [9]. The researchers chose to implement a more sophisticated than SIFT but also slower algorithm - ASIFT. They claim to achieve up to 70 times speed-up in compare with the original ASIFT implementation. However, the implementation of GPU-ASIFT is not publicly available, whereas SiftGPU is free to download and use with the full source code provided.

2.2.2 Features matching

Features matching (also known as keypoints matching [22]) is a step of finding the same features in different images after the feature extraction step. Matching features are found by searching for the nearest neighbours in the features database [22]. Approximate algorithms are used instead of exhaustive search to speed up the process. As with the SIFT feature extraction algorithms, there are also various algorithms and implementations proposed for the feature matching. In his Photo Tourism project, N. Snavely [29] uses a straight forward matching approach while C. Wu uses a multi-threaded

[2]http://www.opengl.org/documentation/glsl/

GPU accelerated feature matcher that is included in SfitGPU software [33]. Building a database of the matching features is necessary before estimating camera parameters and 3D point locations.

2.2.3 Bundle Adjustment

Estimating camera parameters and locations of the 3D points requires solving non-linear optimisation problems. Bundle adjustment refers to adjusting a "bundle" of light rays that intersect the features in the images. In other words, bundle adjustment tries to minimise the measurement and projection errors in the whole model [30]. In order to reduce the size of the problem, researchers [29, 34] use an incremental approach. Firstly, they estimate the parameters of the initial pair of cameras. Later, other cameras are added one at a time and the parameters are estimated for the each new camera. According to [34] the complexity of estimating camera parameters using an incremental approach reduces from $O(n^4)$ to $O(n)$. Thus, the increased speed of the bundle adjustment step allows reconstructing models from huge datasets of pictures. As presented in [34], SfM algorithm was able to rebuild a part of the Central Rome from the dataset containing more than 32 thousand pictures in less than 2 hours. The high performance of the algorithm was also achieved due to a multi-core implementation of the bundle adjustment step [35]. The algorithm can run on multi-core Central Processing Units (CPU) as well as multi-core GPUs yielding the algorithm speed improvements to up to 30 times. The source code and implementation of the Multicore Bundle Adjustment are freely available[3].

2.2.4 Dense clouds and meshes

Usually, after the described steps, SfM software produces a sparse point cloud. Initially, such point cloud does not have enough points to make a clear view of the object that was reconstructed. Nevertheless, the sparse point cloud can be further processed to add more points to the cloud. A Path-Based Multi View Stereo (PMVS) algorithm can be used to create a dense point cloud [15]. Although the running time of the reconstruction increases, as an additional step of dense reconstruction is added, the models after the reconstruction look fuller and are better looking than the models created only from sparse point clouds. In order to make the dense reconstruction process more scalable, Y. Furukawa presented Clustering Views for Multi-view Stereo (CMVS) algorithm[4]. The algorithm prepares input data for the

[3]http://grail.cs.washington.edu/projects/mcba/

[4]http://www.di.ens.fr/cmvs/

PMVS by grouping similar images to clusters of manageable size. PMVS runs with input from the each cluster and the produced results are merged to the final result. In this way, dense reconstruction can be executed in a multi-processor environment, greatly enhancing the speed of reconstruction. The implementations and source code of PMVS and CMVS are free to use[5].

Additionally SfM algorithms can produce readymade meshes using Poisson Surface Reconstruction [17] or Schematic Surface Reconstruction [36] methods. Poisson Surface Reconstruction method can reliably reconstruct a mesh from the point cloud if the cloud is dense enough. It is resilient to noise and gaps in the point cloud and works with any unordered dataset. For the sparse or highly noisy point clouds, however, the algorithm cannot produce well looking meshes. Schematic Surface Reconstruction method uses a slightly different technique to reconstruct models. It can also work with sparse point clouds. Its main purpose is to reconstruct architectural models, thus the algorithm is tuned for shapes that are common in architecture such as planes and cylinders. The Schematic Surface Reconstruction algorithm performs better than the Poisson Surface Reconstruction algorithm in reconstructing buildings or parts of the buildings. However, the implementation of the Schematic Surface Reconstruction algorithm is not publicly available. Therefore, most of the SfM software uses the Poisson Surface Reconstruction algorithm for generating meshes out of point clouds.

2.2.5 SfM Software

Many tools were created for generating point clouds out of 2D images data. Some solutions are commercial while others are open sourced and free to use. Proprietary solutions include Microsoft Photosynth[6], Autodesk 123D Catch[7] and 3DF Zephyr from 3D Flow[8].

Proprietary SfM solutions offer good quality reconstructions. Photosynth, 123D Catch and 3DF Zephyr are implemented as web services, thus user only have to upload his or her photos and the computationally expensive 3D reconstruction is done on the server side. Even though web interfaces can be used free of charge, there are limitations on the number of photos and the size of the models that the services produce. Autodesk 123D Catch is also a standalone program that can run in Windows environment. Features offered by these tools also include easy ways to post-process and edit models.

[5]http://www.di.ens.fr/cmvs/
[6]http://photosynth.net/
[7]http://www.123dapp.com/catch
[8]http://www.3dflow.net/3df-zephyr-pro-3d-models-from-photos/

However, since these solutions are not released as free open source solutions, we cannot use them in our research.

Most popular open source solutions are Bundler[9] by Noah Snavely, VisualSFM[10] by Changchang Wu, Insight3D[11] and SFMToolkit[12] which includes tools used in Bundler and VisualSFM solutions. All these tools are free to use but VisualSFM is not a fully open sourced solution. Some parts of the program are closed-sourced, such as the triangulation algorithm which estimates initial positions of the 3D points.

Open sourced SfM software imposes more difficulties in reconstructing models from photos than web service based solutions. Nevertheless, with the correct parameters and adjustments, the open source solutions offer comparable performance of producing acceptable quality models.

Bundler is a command line based tool used to reconstruct models from large collections of unordered images. It uses D.Lowe's SIFT algorithm for the feature extraction stage. Feature extraction and bundle adjustment is done by the Bundler software itself. Furthermore, it can be used with CMVS/PMVS software to create a dense point cloud. Bundler was used in a Photo Tourism project [29] and proved to be capable of reconstructing huge scenes from thousands of unordered images taken from Internet sources such as Flickr or Google Images.

Insight3D has feature-rich Graphical User Interface (GUI) and a large set of functionality. However, the process of the extraction of features, features matching and generating a point cloud is very slow in compare with other tools.

VisualSFM is a free to use software that can build sparse and dense point clouds. Among other features it can also build a triangulated mesh from the point cloud. The point clouds built using VisualSFM contain many points and are visually nearly as good as the ones produced by proprietary software. Furthermore, VisualSFM is a very fast solution capable of exploiting the latest multi-core hardware to speed up the SfM process. The complexity of the SfM algorithm used in VisualSFM is $O(n^2)$ and it is able to reconstruct as many as 2000 points in one second [34]. VisualSFM software is still under active development and new features are constantly added to this software.

SFMToolkit combines some parts of Bundler and VisualSFM to build the point clouds. It also includes Y. Furukawa's algorithms for building dense point clouds. It is a free to use and open sourced project aiming to provide a complete photogrammetry solution in one package.

[9] https://www.cs.cornell.edu/~snavely/bundler/
[10] http://ccwu.me/vsfm/
[11] http://insight3d.sourceforge.net/
[12] http://www.visual-experiments.com/demos/sfmtoolkit/

New researches focus on improving the accuracy and performance of SfM. Sinha et al. [27] proposed a technique that can decrease the time needed to estimate camera parameters by the order of two magnitudes. By the use of vanishing points it can also estimate the positions of the points more robustly and is easily parallelisable. Moulon et al. [24] proposed another global calibration algorithm that is as fast as VisualSFM incremental approach but provides better accuracy and a very low number of false positives. It also removes the problematic step of finding a good initial pair, which is present in the incremental SfM pipelines. The latter research was implemented as a part of OpenMVG project[13]. However, there are still a few issues to resolve in order to adapt the algorithm for a wide use. Currently, the algorithm only accepts images with the same parameters (e.g. photos taken by the same camera and of the same image dimensions in pixels).

In our work, we used VisualSFM software for SfM operations. It proved to be very fast [34] and produces better results when compared with other popular SfM software such as Bundler or Insight3D. VisualSFM uses SiftGPU that is able to run D.Lowe's SIFT algorithm on multiple GPU cores and also benefit from multicore CPU systems. Even though there were many enhancements to the original SIFT algorithm such as CSIFT or ASIFT, the speed of SiftGPU implementation clearly outperforms other implementations of the mentioned feature extraction algorithms. However it is possible to easily replace the feature extraction algorithm in the future if a better implementation of SIFT based algorithm will be released. VisualSFM also comes with an enhanced bundle adjustment library - Parallel Bundle Adjustment (PBA). The library is optimised for multicore systems and can run on Nvidia GPUs that support CUDA[14] technology. Even though, our host machine was equipped with a graphics card that does not support CUDA, PBA was able to run very fast using only CPU parallelism. Another feature, that made VisualSFM suitable for the project is its command line interface (CLI). VisualSFM CLI allows creating models from specified input photos, specifying which image pairs to match while building a sparse point cloud, resuming the creation of a model, geo-referencing a model with specified ground control points. In order to create dense point clouds, VisualSFM uses CMVS/PMVS2 software. It is possible to create a mesh out of generated point clouds using Poisson Reckon algorithm [17]. However, the results of mesh creation were not fine for sparse or incomplete point clouds. There are other methods to generate a mesh from a point cloud such as Schematic Surface Reconstruction [36], but it turned out that for our work, we do not need to recreate a mesh from

[13]http://imagine.enpc.fr/~moulonp/openMVG/

[14]http://www.nvidia.com/object/cuda_home_new.html

a point cloud, thus the part of mesh generation from point clouds is irrelevant. VisualSFM is also compatible with other tools such as Bundler, as the program can save results in different file formats.

2.3 Navigation

Current navigation systems are able to find a path between any two (or more) given points. The path can be the shortest or of the least weight (e.g. the fastest). Of course, the navigation algorithm needs to have a complete knowledge about the map where it is searching for the path. The maps and obstacles (not navigable areas) can be represented in different ways. Simplest maps can be represented as a regular sized grid, the more sophisticated ones can be of any shape and difficulty e.g. presented as a triangulated mesh. In either case, a navigation algorithm should be robust and faster than a brute-force approach of finding the correct path through the map.

The most known algorithm for path finding is Dijkstra's algorithm [28]. It is simple and was already proven to work in many applications. It can find a shortest path from the source to any destination in a navigation map. However, in order to increase the performance of path finding, especially in computing intensive applications such as computer games, researchers developed a new path finding algorithm called A* (A star)[15]. It is knowledge and heuristic based algorithm that uses similar to a breadth first search approach. The algorithm requires a cost function to determine the costs of each step (or an edge between two graph points). Then, for the next step it picks the node with the best estimation (e.g. the least distance), while keeping other paths in a sorted queue. Basically, it uses the knowledge of already travelled distance and the estimated distance to neighbouring nodes. The A* algorithm has the worst case performance of $O(E)$, whereas Dijkstra's algorithm runs in $O(E + Vlog(V))$ time (here E is the number of graph edges, V is the number of nodes in the graph).

It was proven to be faster than Dijkstra's algorithm and many of its variations are currently used in navigation systems and computer games. Researchers proposed ways for the algorithm to work not only with regular sized grids or weighted graphs but also with triangulated meshes [10]

2.3.1 Navigation map

The difficult part of navigation system is constructing a navigation map. Navigation systems use various map representation techniques [13]. Kim et

[15] http://en.wikipedia.org/wiki/A*_search_algorithm

al. [19] used a special strategy to place marks in the area thus they had the map of the area before hand. Irschara et al. [16] did build the model of the area but they did not attempt to mark obstacles and distinguish navigable paths.

Becker et al. proposed a modelling of building interiors technique based on combined grammar [5]. They use the fact that indoor environments are usually of a predictable shape: corridors have long straight walls and offices are of somewhat rectangular shape; different rooms cannot be overlapping but can be adjacent and perfectly aligned. Such initial knowledge was used to build a grammar and supplement an automatic building of the indoor environments from sensor data. In the research, the researchers used traces from foot mounted sensors and were able to come up with reasonably good results. However, the system was not able to detect small obstacles that may be present in navigation paths such as tables, cabinets or indoor plants. Moreover, it cannot work with irregular shaped indoor environments.

Chapter 3

System Overview

This chapter gives an overview of the image based navigation system. It describes the functional and non-functional requirements as well as architecture of the system. The client – server interaction and requests/responses used in the system are described last.

The functional requirements are trivial for most common navigation systems. Basically, a system must help the user to get from one place to another. However, our system has a very important non-functional requirement - work in the indoor environments without a requirement of infrastructure. First of all, that means, GPS cannot be used and another type of positioning technique must be implemented to locate a current position of the user. Due to a requirement of high accuracy, the indoor positioning methods such as using mobile cell ID or dead reckoning are discarded as being not accurate enough. Moreover, we want to use no infrastructure, thus Near Field Communication (NFC) or Bluetooth tag based indoor positioning solutions as well as WiFi based ones are also not suitable for our system. Therefore, we decided to use computer vision based localisation techniques and adjusted the functional requirements regarding the selected technology.

3.1 System requirements

Image based indoor navigation system navigates the users in indoor areas such as libraries, shopping malls or offices. It has the following set of functional requirements:

- Generate a point cloud from input images. The input is smartphone photos taken in an area. The algorithm should reconstruct a point cloud from the images which will later be used for extracting a navigation mesh and localising a position of a user.

- Update the point cloud model by growing it. The system should append new photos to an existing model to make it larger and allow a modular creation of the area.
- Generate a navigation map from the point cloud. The system has to generate a mesh suitable for navigation in the area. It should extract the obstacles and distinguish between traversable and non-traversable areas and output a navigation mesh that can later be used for path finding.
- Locate a current position of a user based on a supplied photo. The user takes a photo in an area and uploads it to the system. The system should estimate the position of the photo in a previously built point cloud and return the 3D position and orientation.
- Allow choosing a desired destination by either taking another photo or picking one from already predefined locations.
- Plan a path between the current position and the destination in a generated navigation mesh. The path should include a turn-by-turn list as well as visual information from the previously generated model to guide the user.
- Guide the user through the path using visual and textual information. The system should generate instructions for the each turning point in the planned path. An instruction includes a direction where to turn, estimated distance and an image showing the surroundings near the turning point. Instructions must be shown to the user on her or his smart device.

Furthermore, the system also has important non-functional requirements:

- Work in infrastructure-less environments
- Locate the position of a user within a few meters accuracy
- Work with widely available consumer devices (e.g. smartphones or wearable devices).

3.1.1 Example scenario

On a sunny Sunday morning, a person A goes to a newly opened shopping mall with his family. The mall is full of all kinds of shops, cafes and restaurants that are located over several floors. A single Sunday would not be

enough to visit all the places in the mall. Even finding that shop, which sells household equipment, may be tricky, without knowing which way to go. Fortunately, after unsuccessfully trying to find an easy way to reach a shop in the information stand, A downloads and application for indoor navigation in the mall. He takes a photo using his smartphone of the surroundings and chooses a shop where he wants to go. In a few seconds he is following the path shown on his mobile device and finds his favourite shop in no time. Since A is not keen on walking through clothing stores (though his wife definitely is), he decides to spend time just enjoying a cup of coffee in one of the cafes. However, when it is time to go home, he needs to firstly find his wife. Now, not only he needs to know his current location but so does his wife. Again, by taking a photo in an area where person A is, and after his wife takes a photo of a place where she is, the handy app shows the shortest path for A to reach his wife. In the end, the family got out of the mall after visiting all the shops they wanted without spending hours of finding the shops or finding each other.

The example described above is only one of the possible applications of the image based indoors navigation system. However, it already shows what the required functionality is. Now to implement such a system, we need to define not only the requirements but also the architecture of the whole system.

3.2 System architecture

After analysing the strengths and weaknesses of other indoor navigation systems, we decided to build our system based on several already existing methods.

We chose to use computer vision based Structure from Motion (SfM) techniques to build a map of the area and to obtain the current position of a user. For navigation, we chose to use A* (A start) algorithm based path planning algorithm implementation.

We decided to implement our navigation system as a client-server model. The motivation behind the design is that SfM operations are compute intensive and the point clouds that are used to represent maps require large storage space. Furthermore, a server can provide data to multiple clients and scale the resources as the number of clients or the size of the point cloud model grow. In our system, certain tasks are assigned to the server and others to the client software.

Figure 3.1 presents an overview of the system architecture. A back-end

server creates 3D models of the indoor areas, locates the user in the created model and generates instructions for navigation. Client software runs on a mobile or a wearable device such as an Android smartphone or Google Glass[1]. The main functions of the client are supplying photos for the reconstruction of the 3D indoor models, uploading images for finding the location of the user and showing navigation hints to the user. Even though both client and server parts contain quite a lot of functionality, the client is considered to be "thin" and most of the data processing is done on the server side.

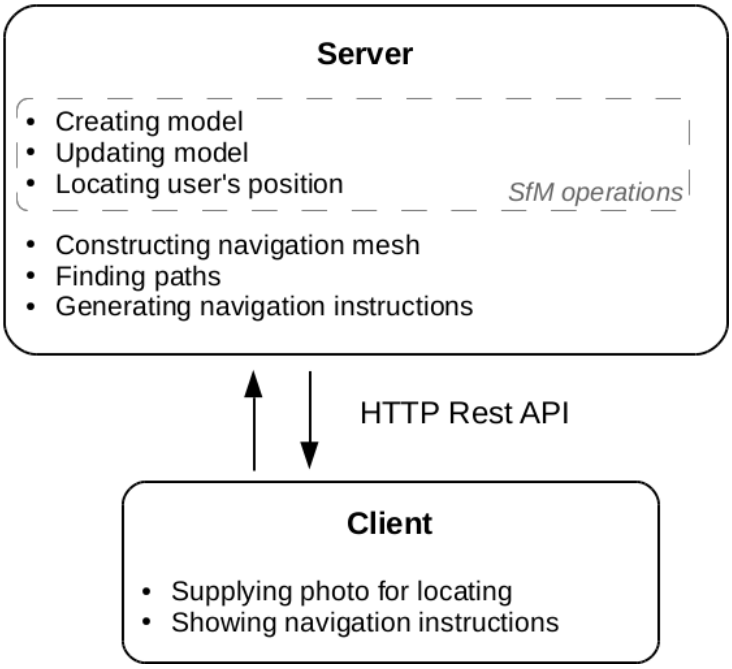

Figure 3.1: High level architecture of the system and its main functions

The functionality of the server can be divided into two main parts: long running operations and real time functions. First of all, the server must be able to create an area model. For the task, it needs to process a number of input photos. After the 3D model is generated, the server has to extract obstacle information from the model and generate a navigation mesh. SfM model creation is a resource demanding operation, but since it does not need to run in real time, it can be executed when the server load is low. Real time operations include localisation of a user and navigation instructions generation. It is important to minimise the latency to localise a position by

[1]http://www.google.com/glass/start/

the supplied photo and to calculate a navigation path in the model between the start and destination positions.

3.3 Client – Server interaction

During the startup, the server initialises a model of an area and its navigation mesh. After that, it handles client requests in respect to the initialised model. The requests and responses of the server are presented in Table 3.1. For the prototype implementation, the requests only include navigation related functionality, assuming the model was already built before running the server. After the server is up and running, client applications can connect and issue localisation and navigation queries.

The main purpose of the client application is to navigate a user in an indoors area. The user takes a photo of any place in the area and uploads it to the navigation system server. After uploading the photo for localisation the client receives a located position. The client application requests for an overview minimap image and displays it to the user with the located position in the map. When a user chooses the destination by either uploading and localising a position of another photo or by choosing a destination from predefined ones, the client application requests the server to find a navigation path between the start and the destination. The client application then receives navigation hints from the server. Hints are shown to the user on the screen. They include turn-by-turn directions and some of the pictures that were used to create the 3D model. Those images show objects at the turning points and help the user to easier find a way to the destination. During the navigation, the client program can constantly take photos (e.g. every 60 seconds) to update the navigation path and to make sure the user is following the path. The client application is suitable to be implemented as a smartphone application, since a typical smartphone has a well enough camera to supply image data for localisation, has a network connection to reach the server and a big enough screen to show navigation instructions.

In order to limit the amount of state in the server we chose to use a Representational State Transfer (REST) server design. It uses Hypertext Transfer Protocol (HTTP) to communicate with the client. Requests and responses are sent using JavaScript Object Notion (JSON) objects. If a request includes image data, it is sent as a HTTP multipart request. If a response includes an image, it is sent as a byte payload in a simple HTTP response. We used the Spring framework[2] to implement communication protocol on the

[2]http://www.spring.io

Request	Parameters	Response	Description
/location	Image bytes	3D location of image	Returns the 3D location of the supplied image or null if the position is not found. Response is returned in JSON format
/route	Start and destination coordinates	Route object with the positions of turning points and instructions	The server plans a route from the start to the destination. The route includes all the turning points between the start and the destination as well as the list of instructions. Response is returned in JSON format
/image	Image ID	A JPEG image	An image ID is the one specified in an instruction. This request is used to asynchronously fetch instruction images from the server
/minimap	minimap ID	A PNG image	Minimap specified by the route response. This request is used to asynchronously fetch the minimap image for a planned route from the server

Table 3.1: Description of the server requests and responses

server side. The framework provides an easy way of handling requests and responses supports many types of messages and runs on a Java Virtual Ma-

chine (JVM). The Spring framework can be used on both client and server, thus making it easier to build a compatible client.

In a simplified version of the communication protocol, it would be enough just to return all the routing instructions (including images) in one request/response as opposed to the current solution (see Table 3.1). However, then the server would have to resolve both start and destination, each time any one of them changes. Since the finding of a location is an expensive operation it is better to let the client resolve both positions before hand. Then, if the client wants to change the destination, in order to plan a new route, only the new destination position would have to be resolved. Furthermore, if a client wants to navigate to an already known object (with a fixed position) there is no need to resolve the destination and the coordinates can be supplied instantly.

A decision to exclude actual image data from the route response was made to optimise the performance of the client. All the images of navigation instructions are not going to be shown instantly and at the same time. While the first one is shown (or the minimap), the other images can be preloaded on demand in the background. Thus, minimising the time needed to prepare the first instruction and to start the navigation process on the client side.

Chapter 4

Creating SfM Model

The first step towards an image based indoors navigation is to create the navigable map - a 3D model of the area. Structure from Motion (SfM) models can be represented as sparse or dense point clouds. Creating a dense point cloud takes significantly longer time than creating a sparse one, since the dense cloud is built from the sparse point cloud.

Another important task for a navigation system is to be able to locate a user in the map. Furthermore, the system must have knowledge of the area to guide the user. A navigation map used in our navigation system is obtained from a sparse point cloud produced by Structure from Motion (SfM) software. Furthermore, SfM techniques are used to obtain the current user position and to mark his or her destination in the SfM model.

The general pipeline of creating a sparse and dense models with VisualSFM is presented in Figure 4.1. Basically, the input is a set of digital photos and the output is a 3D point cloud that represents an SfM model.

Header	Name	Version	Nr. features	0x0005	0x0080
Location Data	x	y	colour	scale	orientation
	...				
Descriptor Data	128 unsigned char matrix				
	...				
EOF					

Table 4.1: Binary *.sift* file format. Each cell represents a 32 bit field

The software internally uses SiftGPU [33] to extract the features from the input images. Extracted features are stored in binary *.sift* files. The layout of the files is presented in Table 4.1. Each feature is described as a 128 bit matrix and 160 bit feature location information. The *.sift* files are used later on in the features matching and point cloud constructing steps.

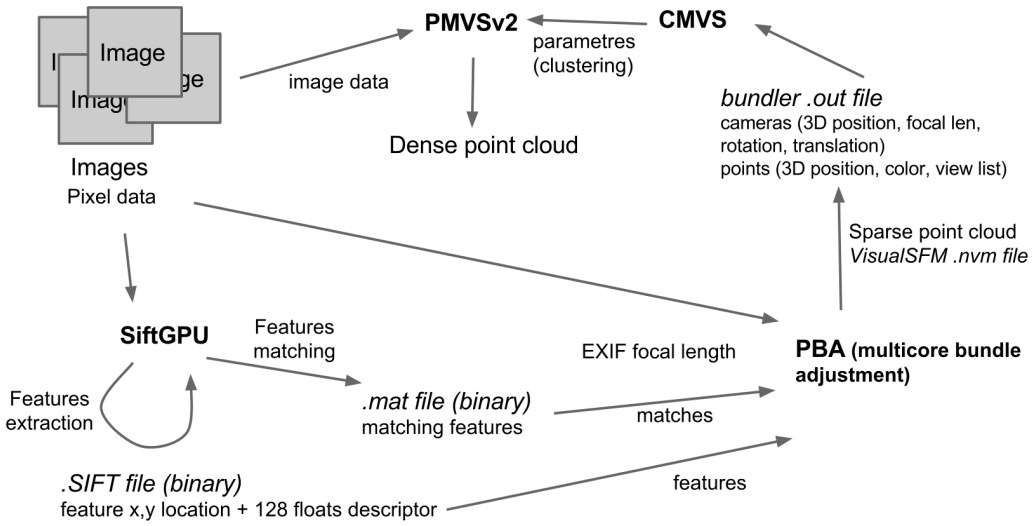

Figure 4.1: VisualSFM pipeline of creating sparse and dense point clouds

Features matching step is also performed by the SiftGPU software. This step produces another kind of binary files, each file storing the information about which same features are also found in other images. The files are in *.mat* file format that is non-standard[1] and only used by VisualSFM and SiftGPU. *.mat* files store mapping between the same features in the current image and the images with which the current image was matched. Both features and features matches files are associated with the original image from which the information was extracted. Thus, both file names start with the image name (without the file name extension). In the reconstruction step, VisualSFM finds the *.sift* and *.mat* files by the names of the supplied input photos. In this case, if a photo was previously used in a reconstruction, its features are already extracted and the extraction step for the image can be skipped. However, matching features file will be extended by appending information from other input images.

Before running the multicore bundle adjustment step (PBA), VisualSFM estimates camera parameters such as camera focal length in pixels and radial distortion [32] and estimates a rough triangulation. Camera parameters are extracted from each input image and represent a camera that was used to take the image. The software uses information from the JPEG EXIF header to determine the focal length of the camera lens. Otherwise, the default value

[1]https://groups.google.com/forum/#!topic/vsfm/NMx2ubzakhA

is used. During the triangulation, the software tries to estimate positions of 3D points and initial camera transformation matrices. Since VisualSFM is a closed source program, not all the details are known about how the triangulation is done and how the parameters for the PBA are computed.

Parallel bundle adjustment reconstructs the 3D model using camera parameters, calculated feature matches and pixel data from images (for colour information). A calculated point includes 3D location, point colour and a list of images where the particular point is seen and which features were used to estimate the point. After the model is constructed, it is saved in an *.nvm* file. The file contains a list of cameras with their 3D locations, rotations and focal lengths and a list of 3D points where each point has the mentioned parameters. It is also possible to save a model in a different file format such as Bundler *.out* file.

The ability to save a sparse point cloud to a Bundler *.out* file allows using CMVS/PMVS2 software to create a dense point cloud. CMVS prepares the required parameters and clusters the model into smaller parts thus allowing PMVS to run in parallel on different parts. After the PMVS algorithm finishes, the results are merged into one dense point cloud. This step also requires images pixel data information. The created cloud is stored into Stanford *.ply* file format[2]. Most of the 3D editing software such as MeshLab, Blender or Maya supports the format and can be used to further modify or view the dense cloud.

In order to adapt the functionality of our selected SfM tool (VisualSFM) to our needs, we created a software solution that wraps VisualSFM and uses it in CLI mode as well as include other functions used for navigation. More details about the created software are presented in the chapter 6.

4.1 Performance issues of creating 3D models

It turned out that a sparse point cloud model is sufficient to locate a user. The observation removes the need to create a dense cloud thus improving the performance of the system.

Creating a point cloud is a computationally expensive operation. A model reconstruction from 262 images took more than 100 minutes to finish on our development machine (using GPU acceleration and 4 CPU cores). The created model represented just a part of Aalto Computer Science building library (300 m^2). To fully recreate a model of the library more than twice as many photos could be needed. For bigger areas such as a shopping mall,

[2]http://paulbourke.net/dataformats/ply/

thousands of photos must be used to fully cover all places and obtain a sparse reconstruction of the building.

SfM software (VisualSFM in our case) is able to work with any kind of unsorted collections of photos. Of course, if the photos are very different or of "bad quality", they may not be included in the model and will not contribute to the point cloud. Thus, the computation capacity used to process such photos will be wasted. During the Thesis, many different approaches were tested on how to take "good quality" photos that would maximise the contribution to the point cloud and minimise the number of images needed to rebuild the model. Some observations that were made during photo collection stage are described in the chapter 8.

Another factor that influences the time needed to build a model is the input sequence of the images. Default behaviour of VisualSFM is to match the features from each image to the features of all other images. Such behaviour requires to match n^2 image pairs (n is the number of input images). It turns out that the matching part is the most time consuming part of the reconstruction. For the example set of 262 photos of the library it accounted for roughly 94% of the time (5807 seconds out of 6185). Nevertheless, we developed a way to increase the matching speed by decreasing the number of matches needed to build a model.

The photos that are taken from a different angle (e.g. one is facing a northern wall of a room and the other one - southern) do not share any common features thus there is no need to match them. Since VisualSFM cannot distinguish image locations or rotations beforehand, it does a full n^2 matching. However, there is an option to specify which images should be matched before the reconstruction. If we know that pictures are taken in a sequence with a small relative rotation and translation between each other or extracted from the video footage, there is no need to do a full matching where each picture is matched with every other picture. Instead, they can be matched in a sequence, i.e. picture N matched with picture N + 1, N + 1 with N + 2 and so on. In order to have a more robust matching, a picture should be matched with more than one subsequent image. The number of how many subsequent pictures should be matched is a trade-off between accuracy and speed. The more pictures are matched, the better model is produced. In cases when there are not enough matches, a model might not be produced at all. Thus, the threshold should be considered wisely to achieve close to optimal speed and accuracy. It should be noted that for the images taken in different areas and/or using different cameras the number of subsequent matches can be very different. During our experiments, matching each image with 10 subsequent pictures produced connected and fairly good point clouds. When the matching was done using more than 20 subsequent pictures, the results

were very close to the ones produced using a full matching. The results of using different numbers of subsequent matches for the sample of 262 photos are presented in Table 4.2. The results show that using 25 subsequent images to match produces same result as using a full match but reduces the time needed for matching almost 6 times. Of course, the produced model had fewer points in a sparse point cloud than the fully matched model. However the difference is only around 9%, thus a small trade-off in the quality of the model yields high savings in processing time. The "Produced models" column shows how many models were produced from the input images. The number actually shows how much fragmented the model is. Ideally, the SfM software should produce only one model since all the photos were taken in a same area. However, due to poor coverage of some areas or not enough matches between pictures, the model becomes disjoint. In this case, there is a lack of photos at some point, since even after the full sequence matching VisualSFM was unable to produce an unfragmented model. The model merging techniques described in the later sections allow merging several smaller models to one and help to solve fragmented model problems.

Subsequent Matches	Matching time (s)	Produced models (1 - optimal)	Generated points
10	590	5	13967
20	858	3	23823
25	1080	2	24948
30	1259	2	26658
Full matching	5807	2	27233

Table 4.2: Feature matching time improvement using different subsequent matches number

For our model creation algorithm we decided to choose a sequence matching of 15 neighbouring photos. The decision is further justified by the tests described in the Evaluation section.

4.2 Creating models from videos

An SfM model can be created from sequences of photos as well as from videos. Since a video is technically a sequence of many images, SfM software can be used to create point clouds from the frames of a video. We tested the approach to see if taking videos instead of dozens of photos would benefit model creation.

Firstly, the frames have to be extracted from the video. We used FFmpeg library[3] to extract every 5^{th} frame from the recorded video. Since videos are often recorded in 25 to 30 frames per second mode, extracting every 5^{th} frame reduces the amount of images used in reconstruction. It does not make a worse point cloud since neighbouring frames are very similar to each other, considering the fact that one cannot turn a camera very quickly in a time between 5 frames are captured. In this way, photos are guaranteed to be in a sequence, thus the model generation can highly benefit from a sequence based matching described earlier. It turned out that it is enough to match as few as 5 subsequent images to get a connected model out of video frames. However, the extracted frames do not contain enough information in their EXIF tags. The information about the focal length is missing. Focal length is used by VisualSFM to estimate camera parameters. Therefore, we used Exiftool[4] to add the missing information to the extracted images.

However, there are several disadvantages of using video frames. Firstly, there is another parameter to consider: how many frames to take from the video, or in other words, which frames to skip. Using as many as possible may produce better point clouds but it highly increases the time needed to create a model. Skipping too many frames may lead to a failed reconstruction or a highly fragmented one, when instead of one model of the area many small models are created. Secondly, the video may contain lots of blurry frames, especially when the camera is being moved or turned. In a moving picture it is impossible to spot individual frames, and humans cannot distinguish blurry frames. But a single stop-frame may be blurry and hardly recognisable. Even though the feature extraction algorithms can tolerate blur to some extent, usually blurry frames are either discarded by SfM or introduce a lot of noise in the model. Removing all the blurry frames manually would reduce both reconstruction time and noise level. On the other hand, such manual removal would require lots of time and is not error proof since a person has to decide whether a frame is blurry or not.

Furthermore, considering that current widely used devices for recording videos are smartphones and cheap portable cameras, the size of the images extracted from the videos is smaller than the size of most common photos. A decent smartphone can take a photo of up to 8 mega-pixels (8MP) (3264x2448) but a recorded video has the maximum size of 2MP (1920x1080). Even though SIFT algorithm needs more time to process higher resolution images, it produces more features, thus resulting in more points and better point clouds.

[3]http://www.ffmpeg.org/

[4]http://www.sno.phy.queensu.ca/~phil/exiftool/

Finally, the biggest problem was encountered during the reconstruction of the model from video frames. The video was taken in the same Aalto library, covering roughly the same areas as the previous photoset of 262 library photos. More than 1000 frames were extracted from the video and we used Exiftool to add the missing EXIF information. Even though the generated point cloud contained around 44000 points (which is significantly more than in the point clouds made from photos; see Table 4.2), the model was not constructed correctly. Most of the points were lying on a same 3D plane, and the estimated camera positions were wrong. The failure of creating a model was caused by all the problems described earlier.

The problems of blurry and low resolution frames can be solved by using a professional video camera, which would also increase the chance of successfully building a model. However, in our project, we aim to create a system that can work with input from simple consumer devices. Therefore, the preferred way of input data is taking single pictures rather than videos.

4.3 Merging point clouds

In order to build a good and large model of the area one may need to take thousands of photos. The requirement for that many photos imposes several problems. Firstly, it is time consuming to take all the pictures at once and build a whole model. Moreover, it is almost impossible to determine beforehand if the number of pictures was sufficient and the built model would cover all the needed places. If at some place the model is too sparse, then more pictures of that area should be taken to increase the number of points in the point cloud. Secondly, there may be a need to expand the current model, e.g. when a new shop is opened in a shopping mall. However, rebuilding a whole model each time when several new photos are added is a time consuming task. Finally, when dealing with large inputs, the SfM software sometimes produces multiple models form the supplied photos. In this way models should be merged to make one continuous model suitable for our needs. Therefore, there needs to be a way to build a model incrementally by adding new photos to the already existing one and to merge multiple models produced by SfM software, in our case - VisualSFM.

Since VisualSFM is an incremental SfM tool, it adds one photo to a model at a time. One option to grow a model is to load the existing point cloud and camera parameters and add a new photo. Even though such an approach works, it imposes a high time penalty when the number of photos increases. This is due to the time needed to find a position of a new photo. There must be a possibility to add any kind of photo to an existing model, thus a new

photo must be matched against all the existing photos to find its position in the model. A more efficient way is to add multiple photos at once. If we know that all the new pictures were taken in the same place, it is enough to find a position for only one of them. After successfully finding the position, other new photos can be matched with the images of the old model in the found position. In order to make sure, that the new photos are representing the same location, we can build a small model out of the new photos first and then consider only those new photos that belong to the model. Of course, the number of photos and the size of the model should not be too large. In such case there would be a problem to decide, which new photo to choose to start matching. The described approach solves a problem of growing the model with the new photos but cannot help to merge the fragmented models produced by VisualSFM.

The Point Cloud Library (PCL) offers functionality to merge two point clouds. Such operation is called registration of point clouds. However, the point clouds must be very similar and overlap with most of the region. Furthermore, the point clouds must be of a similar size. After testing with our input point clouds, the algorithm could not produce robust results, thus the option to use PCL cloud registration was discarded. Similar registration functionality is offered by MeshLab[5] and CloudCompare[6] software. However, they both suffer from the same issues as the PCL library registration functions. An alternative approach would be to find the same points in the two point clouds and compute the transformation matrix for one of the point clouds. It can also be done by geo-referencing both, existing and new point clouds (see section 5.3). However, it is not easy to tell which points are the same in the clouds. For merging fragmented VisualSFM models, it would be possible to use camera points, since each camera is uniquely identified by a name. After several experiments we found out that not always the fragmented models have common cameras. and there must be at least 4 common cameras to reliably estimate a transformation matrix. Thus, we cannot use such an approach to merge the point clouds.

4.3.1 Reliable point cloud merging with SfM

Our approach is based on the idea of merging small models, obtained from new photos, to an existing larger model. However, we want no limit on the new model size, thus additional techniques are used to merge the two (old and new) models. We decided to use no additional third party tools and rely

[5]http://meshlab.sourceforge.net/
[6]http://www.cloudcompare.org/

on VisualSFM functionality. Our algorithm for adding new photos to a model has several steps. First of all, a SfM model is built from the photos. If a number of photos is less than 3, the model cannot be built (since VisualSFM requires a point to be seen from at least 3 positions to be registered). In such situation, photos are added one by one to the model as a simple incremental approach. If a user supplies 3 or more photos, SfM algorithm is executed and the model is built. Since VisualSFM stores features and feature matches files on the external storage, these files can be reused while merging the new model to the existing one. Secondly, after the new "small" model is built, the same pictures with their features and matches files are added to the existing "large" model. Now we only need to find well matching pair of photos from both models and go to bundle adjustment step. The bundle adjustment step would incrementally add the new photos to the existing model, while skipping the expensive feature extraction and matching steps.

The challenge is to find a well matching pair from the both models without matching every added photo to all the existing ones. In many cases it is enough to match only one of the newly added photos to the existing ones, since the other new images are already "connected" to this one when the model was built. However, in order to get more robust results and to remove a limit of adding only a small number of very closely located photos, more than one of the new photos should be matched.

Firstly, our algorithm takes 20 photos from the existing model. Photos are taken in a way that they are uniformly distributed throughout the model. For example, if the model contains 100 photos, every 5th photo is taken into consideration.

Secondly, the algorithm takes 10 or more photos from the newly created model. Of course, if there are less than 10 photos, all of them are included. Ten or more photos are taken due to a custom selection algorithm. We used an approach which is visually similar to a Breadth-First-Search (BFS) trees. Assuming we have N photos, at each step we take every p = (N / M) photo. Where M is doubled each time: M = 2,4,8 ... N. When M = 2, we take the middle photo, when M = 4, we take the middle, the one at one fourth of N and the one at three fourths of N, and so on. At each step the discovered photos are added to the list. If the list size is less than 10, M is increased, otherwise, the required number of photos is taken and the selection algorithm finishes. Note, that VisualSFM will skip already matched photos, thus even if e.g. the middle photo is taken every time, it does not impose any performance penalty.

Thirdly, the selected 20 and 10 photos are matched against each other. I.e. each photo from the set of 20 is matched with each image from the set of 10. During the matching process, the algorithm gathers the numbers

of successful matches. If a pair has more than 60 common features it is considered as a good enough pair to merge the models. If none of the pairs have at least 60 common features, the next 10 images are taken from the new model and the matching is repeated.

Finally, if there were at least one pair of images with enough common features (matches), the bundle adjustment step is executed to build a new model that contains both old and new models. This model replaces the old model and now includes the newly added photos.

Of course, there are various problems that may arise during the merging of the models. First of all, VisualSFM may not be able to build a model from the added photos. In this case the newly supplied photos are not added and the existing model is not changed. Moreover, there may not be enough matches found during a matching step. It means that an algorithm was unable to find a place where to join the new model, thus the built new model is discarded. Due to unstable fashion of VisualSFM[7], there may be problems of merging two models even when there are enough matches. Therefore, the algorithm checks if the model really increased after adding the new photos. If not, bundle adjustment step can be run up to 3 times on the same data set in order to obtain better results. Re-running the bundle adjustment step imposes only a small time penalty in compare with the matching step.

4.4 Geo referencing point clouds

The process of mapping point cloud points to a real world coordinates is called point cloud geo-referencing. Local point cloud coordinate system can be transformed, for example, into a coordinate system representing GPS latitude and longitude. A new coordinate system can also be relative but expressed in known units e.g. meters. SfM software, in particular our selected tool for sparse point cloud generation - VisualSFM, creates models which are represented in relative coordinates. The points which represent e.g. the floor of the area in the cloud might not be aligned with any of two axes, thus for a further analysis of the model, it needs to be in a defined and known coordinate system. For our navigation system, it is also important to have point clouds in real world coordinates. Not only it helps in estimating navigation paths through the area, geo-referencing also enables to show the user estimated travel distances or travel duration.

Geo-referencing can be done using CloudCompare[8], SfM_georef[9] or Visu-

[7]https://groups.google.com/forum/#!topic/vsfm/BfDRCSiBBww
[8]http://www.danielgm.net/cc/
[9]http://www.lancaster.ac.uk/staff/jamesm/software/sfm_georef.htm

alSFM software. All the solutions can reliably estimate the transformation needed for a point cloud to assign it to a new coordinate system. Geo-referencing is done by assigning new 3D coordinates to a selected point. It is not necessary to geo-reference all the points in the point cloud. Typically, 4 points are enough to estimate a transformation since 3 points define a plane in a 3D space and an additional one defines a face direction of the plane. Of course, the 4^{th} point must not lie on the plane. On the other hand, more geo-referenced points yield smaller errors in estimating the transformation matrix.

Also, the point locations must be reliably measured to prevent further errors. A reliable way of generating point locations for geo-referencing is to use a camera on a tripod. The first photo taken from a location can be marked as an initial point (0, 0, 0). Then, the tripod should be moved a certain distance, e.g. 1 meter, forward and later on to the right. Thus, taking photos from locations (1, 0, 0) and (1, 0, 1). Lastly, the camera must be lifted or lowered to obtain the 4^{th} point - (1, 1, 1). Note, that we chose the coordinate system freely, where x-axis is pointing forward, y-axis is pointing up, and z-axis is pointing right. Also the distance of moving the tripod can be any. After successfully adding the 4 photos to the SfM model, they can be marked with the coordinates obtained during the measurement and the model will be successfully transformed. Another way is to use a photo camera that can tag photos with GPS coordinates. However, the photos have to be taken in an open area. We are focusing on building the indoors environment, but several photos can be taken outdoors and then merged with the model if there are enough visible features of the model from the outside. However, if the indoors model is deeply inside the building and not visible from the outside, such an approach of GPS tagging is not suitable.

For our research, we transformed the initial relative coordinates to meters. We took several pictures after placing a camera in predefined places. Then we marked the first place as a reference point with the coordinates (0, 0, 0). Then, the distances from this point to the other ones were measured and the coordinates of the points were assigned. After creating the model with the referenced photos, we assigned the known positions to the cameras. Finally, VisualSFM recalculated positions of all the points in the model (including the positions of the cameras). We used 7 unique locations and marked 13 cameras - since from some locations more than one photo was taken. Taking multiple photos from a location ensured that the new cameras will be connected to the existing model. Thus, the transformation matrix was estimated using 13 geo-referenced points with the error of 0.025 m.

Geo-referencing is a relatively cheap operation performance wise, in compare to the steps involved in generating point clouds. It is useful, since after

adding new photos to the model or merging another model to the existing one using VisualSFM, the geo-referenced coordinate system is lost and the model has to be geo-referenced again.

4.5 Locating user in the model

The main purpose of creating and storing a model of an area is to be able to locate a user in that area later on. In case of Structure from Motion (SfM), locating a user in an already built SfM model can be done in a few steps. Firstly, a user has to take a photo of a place where he or she is. It does not have to be a photo of a place where the user is standing. It can be a photo of any object that is also visible in the model. This is due to SfM ability to estimate camera parameters, including pose and location. Secondly, the user uploads a photo to the navigation system, where the SfM algorithm is executed using the new photo as an input. When the algorithm finishes, if it was able to successfully estimate the new camera parameters, the user is located, the location information is processed and a feedback is sent to the user.

For finding the pose and location of the new camera we use VisualSFM incremental reconstruction option. In this way, an incremental bundle adjustment is done only on a newly added camera. Also, we specify for which images to calculate feature matches, since we are only interested in comparing the new image with the existing ones and not the existing ones against each other. However, as the model of an area may consist of hundreds of photos, matching each existing photo with the new one yields in hundreds of matching steps. That greatly increases the time needed to locate the user. A simple approach to reduce the time needed for matching is to use selective matching. The idea of the algorithm is the same as in the second step of model merging algorithm described in Section 4.3.1. Here, we also use a Breath First Search tree-like algorithm and monitor the feature matches. However, since only one image is matched, after finding a sufficient number of common features (in our case 60), our algorithm does additional steps to improve the chance of successfully estimating the position of the new photo. Let N be an index of the photo in the model, which has the most matches with the new image. The algorithm also takes photos at the positions N-2, N-1, N+1 and N+2 and also match them to a new photo to get more matches and to estimate the new camera pose more accurately. However, if initially there were much more matches than 60, the step of matching nearby photos is unnecessary.

Our system has several problems to cope with. First of all, it may not be

possible to successfully match some of the images to the model. Such images may be taken in an area, which is not included in the model, they may be too noisy as of moving objects e.g. other people going by and covering most parts of the picture, photos may also be blurry or taken of objects that have not enough features to successfully recover camera parameters. In this case, the searching algorithm runs on all the photos in the model and returns an unsuccessful result. Secondly, locating is a computationally expensive operation. Thus it may not be feasible to try locating a user every one second. Fortunately, since the agent that is guided by our navigation system moves relatively slow (around 1.5 m/s) there is no need to adjust the current position rapidly. Another estimation of the location of the user can be done after she or he moves e.g. 15 meters (or after 10 seconds). Though the continuous tracking of the user is beneficial while navigating, the user has to continuously take photos. It might be a burden using a smartphone, but eventually wearable devices such as smart glasses can take photos automatically without the interaction from the user.

An advantage of SfM based location recognition is that it not only estimates the current position of a user but also the direction to which the user is facing. Therefore, the navigation system does not need to rely on other sensors in a smart device such as electronic compass and can initially tell the user towards which way she or he should turn (e.g left, right, turn around).

Chapter 5

Navigation

All the navigation systems have an algorithm to find a route between any two specified places within a system range. Vehicle navigation systems can find a route between two addresses, GPS based navigations can find a path between two points represented in a geographic coordinate system and some indoor navigation systems can route their users from one office room to another. In order to find a route, a navigation system must provide a way to supply at least the start and the destination positions and have enough knowledge about the area in which the route has to be found. In the vehicle navigation example, the navigation system has a database of available roads and knows that a car can only drive on roads. Thus, it can find a route that a vehicle should take in order to reach the destination. Depending on the information available in the roads database, the navigation can find a fastest or a shortest way, prevent certain roads (e.g. the ones being under construction), plan the best time of taking a journey. Indoors navigation systems, on the other hand, do not need to provide such sophisticated journey planning. The indoors navigation systems are mostly meant to be used by people or autonomous mobile robots. In most cases the shortest path in a building is as well the fastest one due to the constant speed of the agents (people or mobile robots). Therefore, in our navigation system, it is enough to consider only the length of available paths to build the best route.

Navigating users in the indoor environment is very much similar to guiding computer controlled characters in a 3D computer game. There must be an algorithm which finds the shortest path from one point to another in a closed environment. In artificial environments, game engines use either Navigation Graph based approaches or Navigation Mesh based ones [7] to mark the traversable space on the map and to calculate the shortest paths. Either navigation graph or navigation mesh can be used to represent a navigable area. Once the mesh or graph is created, in order to find the shortest

route, various graph search algorithms are used such as Dijkstra's algorithm, Breadth First Search or A* (A star) algorithm. For our system, we chose to use navigation meshes and an A* based path finding.

5.1 Building navigation map

Before applying any path finding algorithm, we need to have enough knowledge about the area where the navigation takes place and build a proper navigation graph or mesh. For our navigation system, one way is to determine the walkable paths in the area, and another way is to find non-walkable areas (obstacles). Clearly, it is always possible to state that obstacles are everything that one gets after subtracting walkable paths from the whole area and the walkable paths can be found by subtracting known obstacles from the area map. For our system, we chose to find the obstacles in the area which were later used to build a navigation mesh.

5.1.1 Navigation graphs and meshes

We might as well build a navigation graph (instead of the mesh) and apply the navigation algorithms on the graph. Navigation graphs have an advantage of less space required to store the map and are easier to use with the path finding algorithms. Due to a straight-forward implementation and light weight, navigation graphs were used in numerous applications and especially computer games.

However, even though navigation graphs were still used in recent computer games such as "Alien Swarm" [8], they were often criticised for inaccuracy and poor performance for wide open areas. Some paths planned by navigation graphs tend to have "robotic" movement and may choose not the shortest route [1]. Navigation graphs, on the other hand, try to solve such problems by presenting a navigable area by a mesh. Meshes are useful for covering wide areas and they can be easily derived from the obstacles. Furthermore, navigation mesh based path planning can be extended to include areas such as bridges and tunnels, thus navigation meshes are suitable to cover non-flat areas such as multi-floor buildings. Since navigation meshes are used in many applications, there are various algorithms for building the meshes and finding shortest paths in them. In the end, we chose to use navigation meshes instead of graphs due to robustness and accuracy of the paths planned using the navigation meshes. Memory overhead for storing the mesh is negligible in compare with the storage space needed by the SfM model, from which the mesh is built.

5.1.2 Obstacles extraction

The first step towards building a navigation mesh is to find the obstacles in the navigable area. Since the generation of the area map is based on Structure from Motion (SfM) techniques, the area is represented as a 3D point cloud. Some of the points may represent obstacles such as tables, chairs and walls while others may represent floor points that are within a navigable area. Point clouds also contain noise and some points may be misplaced. The outlier points from obstacles might be placed on the navigable path thus making it harder to find all the possible traversable area.

To simplify the problem of obstacles extraction from the point cloud, we decided to flatten the point cloud by assigning all the up-facing coordinates to 0, thus placing all points on the ground plane. Since the SfM models are geo-referenced to our selected coordinate system, we know the up-facing axis and can correctly flatten the model. Having all the points on one plane reduces the complexity of the problem, since now we have to solve only in 2D space. However, the height information of the objects is lost during the process of flattening. If the generated area model includes points that lay on the ground level, in a flattened model they will not be distinguishable from the points that represent real obstacles, e.g. walls or cabinets. Therefore, the points at the ground level (or floor points) have to be considered before building the model. They can be removed before flattening the model, thus indicating that a path where the points were removed is traversable. The only problem is to identify which points belong to the ground level. A straightforward way is to mark all points that are below a specified threshold as the floor points. The threshold can be up to e.g. 10 cm to address the noise of the point cloud. If there are stairs or the area has ramps and slopes, however, all the slopes higher than 10 cm from the ground level would be marked as obstacles even though people are able to walk on them. To overcome the problem additional information from the area model should be used. The generated area also includes information about the positions of cameras which were used to take photos for model reconstruction. Considering the fact that photos were taken by humans, it is reasonable to say that the place where the camera was held was not occupied by any obstacle. In this way lowest points around such camera position can be marked as the floor points. Of course, the technique may mark too many points as walkable if the camera which took the picture was placed on the table or very close to an obstacle. This technique also has a radius parameter which determines how many points around the camera should be marked as floor points. Having a very small radius would barely mark any points, while having a too large one could mistakenly mark obstacles as walkable areas.

In our case, we firstly remove the points that are lower than 10 cm above the ground level and the points that are within 25 cm around the camera positions but only those which are below camera. In our model, we did not have any ceiling points. Otherwise, the points that are on the ceiling level at the height of 2 meters or more should be removed.

5.1.2.1 From points to obstacles

Having the 2D points that represent non walkable areas is helpful but the points have to be processed further in order to use them with the navigation algorithms. Before building the navigation mesh we need to extract shapes of obstacles form the points. Since we have unordered points it is not possible to connect each point with an edge thus forming a bounding mesh. The navigable areas inside such mesh would be obscured. It is also not feasible to connect each point with all its nearest neighbours since it would form a complex polygon with lots of edges and will not guarantee an optimal coverage.

One of the solutions to form a shape from points was proposed by Duckham [11]. The solution generates non-convex polygons out of unordered 2D points. The operation is fast and the complexity of the algorithm is $O(nlog(n))$. The algorithm basically wraps the points with a polygon specifying which points belong to the outline and which are the inside points. It can also provide a triangulation to represent all points as a 2D mesh. The algorithm behaviour can be altered by adjusting the *Normalised Length* parameter, which determines the lengths of the edges that will be used in constructing the bounding polygon. If the edges are very long, the "L" shaped obstacles may be presented as a triangle shape instead. It may seem that the most accurate results could be retrieved using the shortest possible length of the edges but due to the noise in the original model, short edges result in staggered obstacles. Smooth surfaces such as walls or shelves look wavy and the extracted shapes are hard to visually recognise. Additionally, the algorithm has an option to set the *Normalised Length* parameter to the "maximum length of the minimum spanning tree". The spanning tree is built from the input points. This option has shown the best results with various inputs.

The algorithm is robust and delivers acceptable results. The only problem is that it always tries to generate a single polygon out of all the supplied points. In our area model, obstacles should be characterised by many polygons, e.g. tables, shelves or chairs should not be connected together to form one large obstacle.

Another option for extracting shapes from unordered points is to use

Computation Geometry Algorithms Library (CGAL). CGAL includes functionality to find alpha shapes [12] from the specified points. It can work with both 2D and 3D points. Though for simplicity and since we already had flattened the points, we used the algorithm with 2D points only. The algorithm can find multiple shapes from the input points. The difficulty is to select a correct alpha value to obtain good results (the maximum value would cover all the points with a single polygon). After testing the algorithm with our data sets, the algorithm was proven to be unsuitable due to problematic selection of the alpha value. Therefore, the algorithm was not as good to extract shapes as the solution proposed by Duckham. Moreover, CGAL is implemented in C++ whereas Duckham's solution is written in Java and can be easily integrated in our project.

Other computer vision libraries such as OpenCV and Point Cloud Library (PCL) have functionality to process 3D point clouds and extract primitive shapes out of the clouds. However, the shapes are limited to cylinders, boxes and plain lines. Also, the algorithms are very sensitive to noise in a point cloud and provide no robust way to extract the shapes. Also, we do not use dictionary or any training steps to find particular objects in a point cloud and mark them as e.g. a table, a chair or a corridor. There may not be enough information in the point cloud and clearly not all the different objects can be recognised. Thus, we chose the approach of extracting obstacles using Duckham's solution.

5.1.3 The algorithm

Here, we describe the mesh building algorithm utilised by our navigation system. It assumes that we already have a flattened point set from a built point cloud with all the ground points filtered out. As mentioned before, the selected Duckham's shape extraction algorithm includes all the supplied points to a shape (i.e. wraps all points with a polygon). In order to use it with our flattened point set we firstly need to extract groups of points that represent different obstacles. In order to do that, we group the points by the distance between their neighbours.

Starting from any point, if its neighbour distance is less than a specified threshold t, the point is added to the group. When all the close neighbours (the ones that are no further than t) of the initial point are added to the group, the newly added neighbour is selected as an initial point and the algorithm continues until there are no more neighbours that would be at a distance less than t. Now, one group of points is formed that will represent an obstacle. The group is removed from the initial point set and a new initial point is chosen. The algorithm continues until the initial point set is empty.

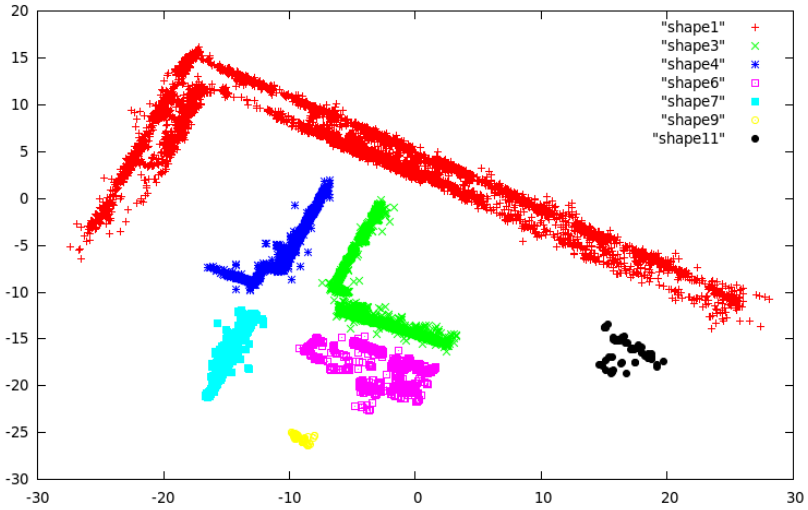

Figure 5.1: Point sets after flattening and running the grouping algorithm

The value of t should be considered as the smallest gap that is still suitable for an agent to pass through. In our algorithm we chose $t = 50cm$. In order to remove the outlier points we removed all groups that were created of less than 10 points. Again, the number of 10 was chosen arbitrarily after a few tests with the current data sets. To get a fast implementation of the algorithm that would work with thousands of points, we used K-D tree implementation from the PCL library to supplement our algorithm. K-D Tree uses fast approximate nearest neighbour search algorithm and is suitable for searching neighbouring points on a 2D plane. The algorithm is implemented in C++ thus providing a high performance and a low memory overhead. The output of the algorithm is presented in Figure 5.1. Different colours represent points from different sets. Note, that the ground points are removed from the point cloud before processing

Shape extraction algorithm processes all the point groups and produces bounding shapes. Those shapes represent unwalkable areas in the map. Shapes are extracted using the Duckham's algorithm and the output is presented in Figure 5.2. The colours of the shapes match the colours of the point groups from which they were extracted. The result was visualised using Blender[1]. As seen in the Figure 5.2, the shapes contain jagged edges. Also, the green shape obscures more area than it should and other shapes look like they are lacking the volume. This is a trade-off in choosing the

[1]http://www.blender.org/

Normalised Length parameter for the shape extraction algorithm. However, the main walking paths are not obscured and the navigable area is presented well.

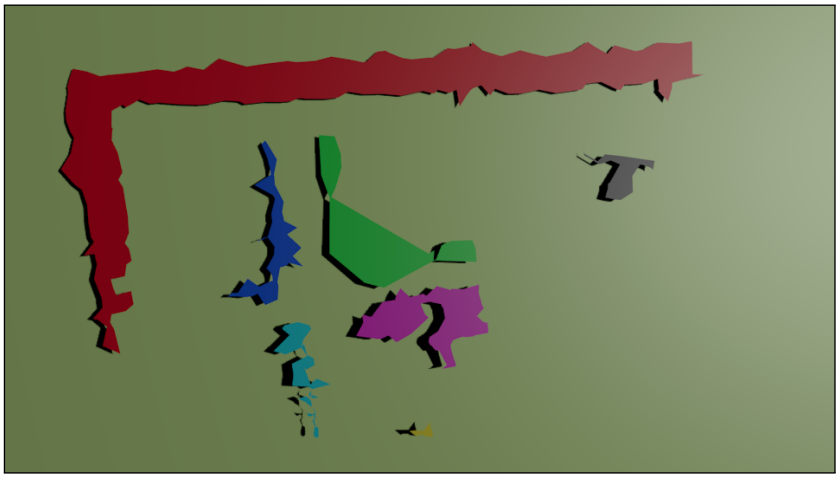

Figure 5.2: Shapes, extracted from the grouped points

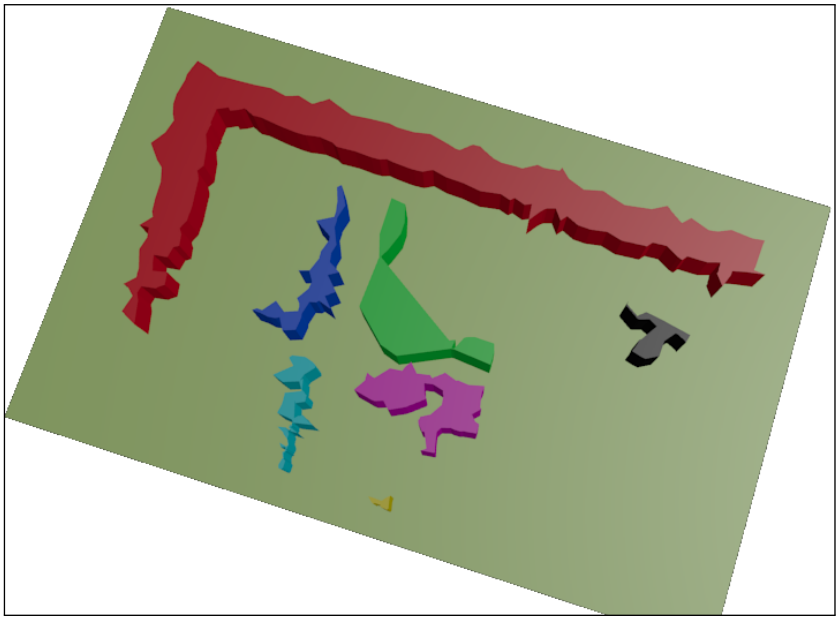

Figure 5.3: A mesh generated from the extracted shapes

After extracting the shape contours, we can use a straightforward ap-

proach to build a navigation mesh. All 2D shapes are extruded towards up-facing axis to form solid meshes and are placed on a 2D ground plane. The created mesh is saved as a Wavefront OBJ file[2]. The output of creating a mesh out of extracted shapes is shown in Figure 5.3. The picture is rotated to show the extruded part of the mesh. The mesh looks like a labyrinth and wraps all the obstacle points. The ground plane represents traversable areas. The mesh can now be used for navigation in the area.

5.2 Finding the path

Once the navigation map of the area is obtained we can find paths from any point in the map to any other point. The most commonly used algorithms for path finding are derived from the A* algorithm or directly uses A*. Initially, A* was designed to be used with grid based maps. However, it can work with triangulated navigation meshes as well, to find the shortest paths by traversing triangles. Since path finding problem was solved in numerous solutions, we decided to adapt already existing solutions rather than creating our own one. RecastDetour library[3] by M. Mononen provides sufficient functionality to find the best paths in a supplied mesh.

The library is considered to be made of two parts: Recast library is responsible for processing the navigation mesh and Detour library finds the paths in the mesh. The project also comes with a demo application which shows how the navigation process works (see Figure 5.4). The demo application shows the calculated traversable areas in blue and the obstacle areas in grey. Start and destination markers are placed on the map. The library calculates a navigation path (shown as a dashed line) between the markers. Also, the demo application shows which triangles were checked while searching for a path (marked in yellow). Since the demo application was successful in finding the paths in our generated mesh, we can use the library for our navigation system.

There are quite a few advantages of using the libraries. Recast is able to adjust the mesh according to the radius of the agent. It means that an agent can have a size and the navigation will not show paths through the areas that are too narrow to walk through. Also, the estimated length of the path is more accurate since it is not aiming to the very corner of an obstacle (the path is adjusted according to the supplied radius of the agent). There are even more parameters such as the height of the agent, maximum climb slope or maximum jump height. Such parameters are relevant for

[2]http://www.fileformat.info/format/wavefrontobj/egff.htm
[3]https://github.com/memononen/recastnavigation

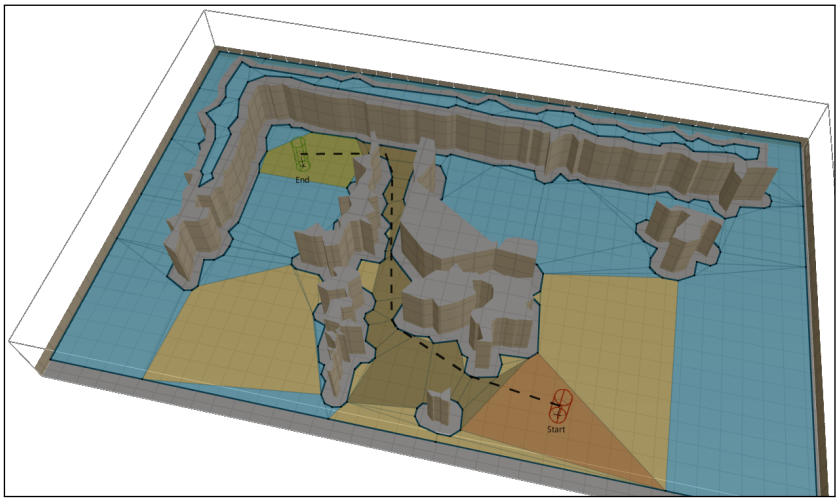

Figure 5.4: Our generated mesh in a RecastDetour demo application

computer games, but for our navigation system, we do not consider climb slopes or jump heights. Our generated area is flat and the obstacles are of the same height. However, if there is a need to include the slope information in the future, the same library would still suit the needs. Detour library is used to find paths in the navigation mesh. Internally it uses an algorithm similar to A* and finds the paths instantly (few milliseconds on our host PC). Processing the mesh by Recast library takes a few hundred milliseconds on our host PC. Nevertheless, it only needs to be processed once, after creating the navigation mesh from the point cloud.

5.3 Generating navigation instructions

After finding the path between user's position and her destination, we need to show the user how to reach the destination. A straightforward way is to show a 2D minimap of an area with the path drawn on it. However, it is not always easy for a person to follow a 2D map, especially of an unknown area. Therefore, we developed an algorithm to additionally generate visual instructions for the each turning point along the path. An instruction includes a photo that was used to create a model of the area, textual information describing where to turn and approximately how many meters to walk until the next turning point. Furthermore, we draw a part of the path on the instruction image to show where the user should walk.

The challenging part is to choose a correct image to show at a turning

point. It should be covering the turning point and be clear enough to understand the surroundings. Our algorithm for finding the correct image works in a following way. Firstly, for a turning point we find a feature that is closest to the 3D turning point obtained from the navigation path. Once the feature is obtained, we select the photos that contain the feature. Secondly, we sort the pictures by distance (the distance between a photo location and the turning point) and take 5 of the closest ones for the next step. Then, we sort the 5 photos by the relative rotation regarding the navigation path. That is, we try to find the photos that were taken facing the closest angle as the user should traverse the path. Thus, the best angle would be facing the same way as the user walks. After sorting, we take the first 3 photos and finally sort them by the distance between the feature and a centre of a picture. The one with the closest centre to the feature (thus also to the turning point) is selected to use for the navigation instruction of the turning point.

After the photo is chosen, we draw a part of the path on the image. We draw an arrow from the previous turning point to the current one (associated with the photo) and another arrow from the current turning point to the next one. The positions of the arrows are found by projecting 3D points back to 2D image. We use camera rotation and position matrices as well as focal length and radial distortion parameters to reliably project a 3D point to a 2D image. Figure 5.5 shows the example instruction images with the drawn path arrows. The current turning point is between the arrows.

Figure 5.5: Instruction images used to show the navigation path to a user. White arrows represent parts of the path that the user should take

Chapter 6

Prototype applications

This section describes the created navigation system prototype. Some pieces of the created software can be used as a service, while others are used as standalone tools. SfM Navigation Tool is an application responsible for creating and growing SfM models and extracting navigation meshes form them. It can also plan the shortest path through the extracted map. The tool is useful for analysing point clouds, since it can visually represent models and camera positions. The Server application acts as a restful service for locating a user in a map, finding a route between two points and providing graphical information about the route. Mobile client application acts as a front end client and contains functions to navigate a user in the area. Most parts of the code were written in Java programming language. Some parts were implemented in C++ programming language in order to be compatible with the used libraries.

6.1 SfM Navigation Tool

SfM Navigation Tool is an application with a Graphical User Environment (GUI) (see Figure 6.1). It has the functionality of creating, growing and georeferencing the SfM models; locating the position of the user in the model; extracting the obstacles from the point cloud and building a navigation mesh; finding a navigation path through the model using the supplied reference photos; analysing point clouds and camera positions. All the steps are visualised and the tool shows how the model changes and how the obstacles are extracted, as well as shows navigation paths and hints. A subset of the tool functionality is used by the Navigation Server to resolve navigation queries and generate navigation instructions. The SfM Navigation Tool is mainly used to create 3D models and extract navigation meshes. It is also utilised

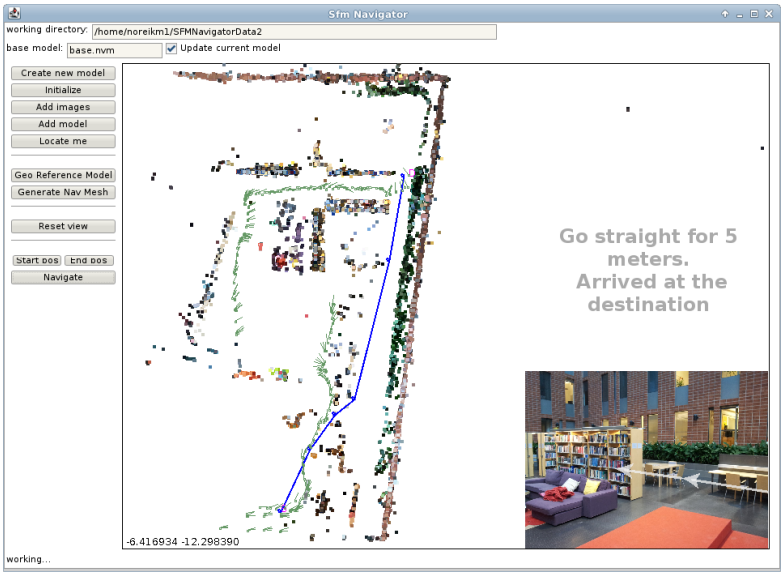

Figure 6.1: SfM Navigation tool showing the created model, navigation path and a navigation instruction

to analyse the point cloud and results of locating camera positions in the model.

6.1.1 Model creation

SfM Navigation Tool works with the files produced by VisualSFM and runs VisualSFM in a command line mode to accomplish certain Structure from Motion (SfM) operations. The tool can create a model from the selected photos. A user specifies a folder where to place the created model files and the photos from which the model will be built. The folder contains special structure, since the created model can later be used for growing and geo-referencing the model, generating navigation mesh, and locating the position of the user. For the creation of the model, the tool copies images to the resource folder and runs VisualSFM sparse model generation. After the model (*.nvm*) file is generated, the tool reads the file and displays the generated model to the user. The user sees the model from the top perspective. The view can be zoomed and panned. It also shows the cameras that were used to take photos and to which way they were facing.

In a case when the created model is fragmented (contains several models), only the first model form the file is taken - VisualSFM puts the largest model in the beginning of the file. Using only one model simplifies geo-referencing,

locating and navigation processes. If, for example, not all the photos were added to the first model, they can be appended to the model using the "Add images" option. Using this option, a user can supply new photos to be appended to the model. The algorithm works in a way described in the section 4.2. The algorithm specifies which photos should be matched and creates a special "pairs" file that will be supplied to the VisualSFM for the model creation. In this case, VisualSFM loads the existing model, loads and adds the specified photos, computes the specified matches and resumes the incremental bundle adjustment. It produces a new *.nvm* model file which replaces the current model file. The SfM Navigation Tool reads the file and displays the updated model to the user. In a similar way, not only the new photos but other already built models can be appended to the existing model.

When the model is built, there is a possibility to locate a user in the model by supplying a photo. If a photo was taken in the area covered by the model, the program finds the place where the photo was taken and marks it on the map. The location of the photo is found by using VisualSFM sparse point cloud building function. The supplied photo is loaded with the model and VisualSFM runs incremental bundle adjustment on that photo. If it is successfully added to the model, the SfM Navigation Tool finds which camera represents the added photo and extracts its position and relative rotation from the model file.

The Tool also has a geo-referencing option. As described in the section 4.3, it supplies known (measured) relative locations to some of the cameras and uses VisualSFM option to geo-reference a point cloud. After the referencing, the model is adjusted to a coordinate system where the up-facing axis is Y, X axis point forward and Z axis - to the right. A user can also see the coordinates of any point in the map by moving the mouse over the map area.

6.1.2 Navigation functions

After the model is geo-referenced, it is possible to generate a navigation mesh and execute navigation based functions (see Figure 6.1). Before starting the navigation process, the navigation mesh has to be built. The generation of the navigation mesh is an operation that includes several steps. Also, it utilises third party Java and C++ libraries to accomplish the task. The details of the algorithm are presented in the section 5.1. The generated mesh is saved as an *.obj* file and the extracted contours of the obstacles are drawn on the map. The path planning library uses the generated mesh to initialise the navigation mesh of the area.

Now the user can supply start and destination locations by choosing the

two photos - one that represents start and another representing a destination. When the both start and destination positions are obtained, the path planning library finds the shortest path between the points. The path is drawn on the map.

A user can also preview the generated instructions on how to travel to the destination. Additionally, the program draws two navigation arrows on the image. First arrow points from the previous turning point to the current one. Another arrow points from the current turning point towards the next one. In this way, a user can see a segment of the path where she or he has to go.

Since the navigation process is executed locally using the model stored on the same machine, it only serves for the testing purposes. In order to make a useful navigation system, the navigation functions have to be reachable by remote client applications.

6.1.3 Statistics

The GUI of the system is useful for showing various statistics. Firstly, it can show the point cloud and camera positions. A user can see 2D positions in real world units of any point of the map by using a mouse. A tool can show the position and direction of located cameras. During the location accuracy tests we located many different photo sets and recorded the located positions to a *.csv* files. The tool reads the *.csv* files and displays relevant info on the GUI. Figure 6.2 a) shows the plotted different photo sets on the point cloud. It also allows configuring plotting parameters. Figure 6.2 b) shows how the tool was utilised to generate a heat map of the areas covered by camera viewing angles. It is useful to determine which parts of the area were covered during the model reconstruction. It is easy to adapt the tool to show also different kinds of statistics, e.g. density of the point cloud. Therefore, the tool can be emplyed by developers to debug and test point cloud related algorithms and extract the statistics from a generated model.

6.2 Server

Server application is utilised to expose the functions of SfM Navigation Tool over the network. The server application can be seen as a wrapper over the existing functionality. It exposes all the functionality needed for localisation and navigation. Model creation and navigation map generation is processed offline, on the server side. Clients can request a server to resolve positions and to generate navigation instructions between two known points. More detailed

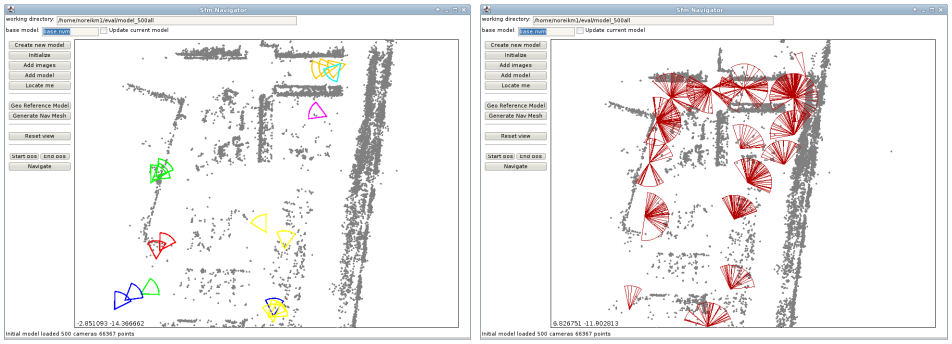

Figure 6.2: Analysis of the point cloud and camera locations using the SfM Navigation Tool. On the left: a) positions and orientations of located photos; On the right: b) heat map of camera viewing angles

description about the technologies and the server requests / responses is presented in the Chapter 3.

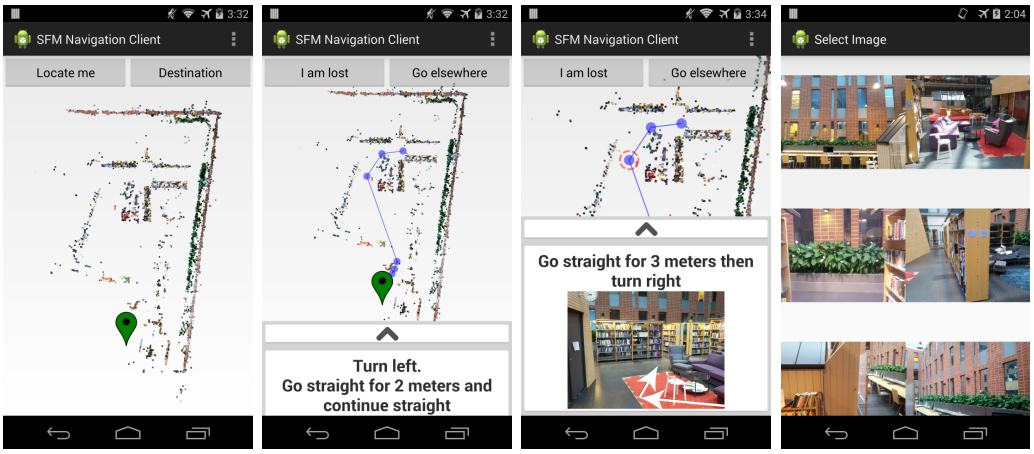

Figure 6.3: Mobile client interface. From left to right: a) Located user's position shown on the mini-map; b) The planned path between user's position and destination; c) The user interface showing an expanded instructions view; d) Photo selection screen after choosing to locate by suplying a photo from predefined images instead of the camera app

6.3 Mobile client

A front-end of our prototype navigation system is a mobile application. The application runs on an Android mobile device. Firstly, the application has to detect a current position of the user. In order to do that, the user taps "locate me" button and takes a picture of her or his surroundings. The picture is sent to the server as a location query. If the photo is located successfully the server returns a 3D position of the photo. The client application then requests the mini-map image from the server and shows 2D top view of the area to the user with the marked current location (see. Figure 6.3 a)). In the prototype client application a destination is also supplied by taking a photo. The prototype application has a set of already taken photos to choose from or allows taking a new photo in an area. After the server resolves a position of the destination photo, the client application requests a route from previously resolved current location to the destination. The server returns a path as a set of points as well as associated images that would help to better guide the user. The user interface now changes to show the results (see Figure 6.3 b)). The recommended path and a first navigation instruction are shown on screen. The user can swipe through the instructions to preview the route. Each turning point (marked as a blue dot) includes textual instructions as well as an image of surroundings (see Figure 6.3 c)). The application also plots the path on the instruction image, thus the user sees what objects he or she should see while going along the navigation path. If a user wants to relocate her position or choose another destination, she can also do that by choosing a photo from a predefined set (see Figure 6.3 d)). Predefined set is useful for testing and evaluation purposes.

If a user is lost or wants to change a destination she or he can use "I am lost" and "Go elsewhere" buttons. The former allows changing the current position by supplying a new current location photo and the latter is used to change the travel destination. After a change in the current location or the destination, the mobile client requests a route update from the server and the updated information is shown to the user.

Chapter 7

Evaluation

This chapter focuses on evaluating the prototype of the created navigation system. Subsections evaluate performance aspects of building Structure from Motion (SfM) models as well as the reliability of the method for building such models. Evaluation highly focuses on the speed of locating the user in a model and finding a path from the current location to a selected destination. Our performance evaluations do not focus on data transmission latency or overhead. The time needed to transmit the request from the client and download the response from the server highly depends on a used transmission medium e.g. WiFi, 3G or LTE. Moreover, since the server mostly performs computationally demanding operations, the time needed to transmit data through the network is considered negligible and is not important in the research.

Photosets that we used during the experiments were taken inside a library which is located in a Computer Science building of Aalto university. The library is a big room full of book shelves, tables and chairs amongst other objects. It is around 25 meters in length and 13 meters width. The environment is rather diverse and contains lots of information for SIFT feature extraction, making it a good environment for SfM model creation. However, it also has glass windows which cannot be detected by the SIFT extractor. Furthermore, the photos were taken at different days and day times, thus in the photos some objects (e.g. chairs) were re-arranged and lighting conditions were different (e.g. sunny days and cloudy days).

7.1 Experiment devices

All the evaluation tests were carried out on the same machine. The machine is a desktop PC equipped with Intel Q6600 quad core CPU (clocked at 2.40

GHz); 4 GB of memory and Radeon HD 4350/4550 GPU with 256 MB of memory. It runs Ubuntu 12.04.4 LTS operating system.

For our system, it is important that a testing machine has not only a multicore CPU but also a fast GPU with the support of OpenGL Shading Language (GLSL) to quickly perform SfM related work. Since we were not building huge models, the size of the main memory was sufficient. However, model creation is resource demanding and requires more memory to build bigger models.

Input images for creating the models were made using a smartphone. The images were photos, taken with Samsung Galaxy S2 (GT-I9100) phone using Camera ZOOM software. Initially, 8 mega pixel (MP) resolution (3264 x 2448 px) photos were taken.

7.2 SfM operations

For positioning and navigation purposes, our navigation system utilises SfM models built using the tool described in Section 6.1. Structure from Motion (SfM) operations are usually resource demanding and time consuming. Furthermore, there are many parameters and factors that affect the speed and performance of the operations. We found out, that for our navigation system two major factors influencing the quality of SfM models and the time needed to build them are the size of the input images and the number of sequence matches.

All the input photos used for this project were taken using a smartphone. Currently, most of the smartphones are capable of taking pictures as large as 8 MP. However, with such images, SIFT algorithm has to process more than 8 million points for the each image during the feature extraction step. A way to decrease the workload is to resize input images to a lower resolution. To test whether downsampling improves the performance without a loss of quality we used a set of 100 photos. The photos were rescaled to 2448 x 1836 px (4.5 MP), 1632 x 1224 px (2 MP) and 816 x 612 px (0.5 MP). We built the SfM model from each photo set (original 3264 x 2448 px and downsampled ones). The results of building a model using different resolution photos are presented in table 7.1.

Not only the feature extraction but also the feature matching time is decreased by using lower resolution photos. Considering the number of points in each created point cloud, 1632 x 1224 px photos provided the best time / points ratio. Interestingly, the model built from photos of 2448 x 1836 px resolution had the most points (almost twice as much as the model built from the original photos) but also took longest time to be processed.

Resolution	Feature extraction (s)	Matching (s)	Total time (s)	Points
816x612	42	166	390	9196
1632x1224	47	189	465	13110
2448x1836	100	800	1377	22819
3264x2448	70	241	598	10777

Table 7.1: Building a model using different resolution images

Seq. match.	Matches	Matching time (s)	Built models	Images in 1st	Points in 1st
5	1280	140	34	33	3460
10	2535	301	17	84	10108
15	3765	442	14	123	14698
20	4970	587	12	121	14556
30	7305	883	11	124	15299
50	11675	1396	9	123	17085

Table 7.2: Model creation using different sequence matching number

In a short conclusion, when input images are downscaled, they become more sharp and SIFT algorithm is able to find more features in a shorter time. Therefore, it was decided to use downscaled photos with dimensions of 1632 x 1224 px.

Also, we carried out several tests to find the optimal number of sequence matches to use while creating an SfM model. As mentioned in Section 4.1, different numbers of sequence matches highly affect the time needed to build a model. However, not only the speed is important but also how well the algorithm generates a point cloud. The results of the tests are presented in table 7.2. The selected test photoset contained 259 photos. As seen in the table, when the number of sequence matches ("Seq. match." column) increases, the number of required image pairs to match and matching time increase as well. Decreasing the number of required matches is preferable but the quality of a model reconstruction is also important. The "Built models" column shows fragmentation of a model. The higher the number, the more a model is fragmented. Ideally, there should only be one model after the reconstruction. Another important column in the table is "Images in 1st". It shows how many images were reconstructed in the largest model. When building and merging models, our system considers the first (largest) model. As the table shows, the optimal number of sequence matches is 15, as it minimises the number of pairs to match and produces comparable results

with the ones obtained using more sequence matches. Therefore, we chose to use 15 images sequence matching for building SfM models to optimise both: model reconstruction speed and quality.

7.2.1 Creating a model

SfM model creation is a resource demanding operation. We tested how fast a model can be created on our test machine by building models from different sized photosets. The pictures to reconstruct a model were taken in a particular way. Basically, starting form a point, several photos were taken facing different angles, and then the camera was moved one step forward. The angles between the photos were around 15 degrees since the photos need to have large overlapping areas in order for SfM algorithms to work. The library area was traversed in different directions. In some places more photos were taken than in the others. Figure 7.1 shows the library model built from 500 photos (flattened point cloud, excluding ground points) and view angles of the photos. The view angles are plotted as arcs. Each arc represents the view that a camera was facing when the picture was taken. For clarity, several camera locations are merged into one arc. Original camera locations are marked as small rectangles. As seen from the Figure 7.1, some places were traversed in both ways, while others only in a single direction. The more cameras were facing a particular area, the more points it had in a point cloud.

We benchmarked the model creation step with photosets of 50, 100, 250, 500, 750 and 1000 photos. The results are presented in figure 7.2. As seen in the results, the time needed to create a model is increasing linearly. It is due to a restricted matching technique of using 15 sequence matches while creating the model. Having 15 sequence matches imposes a requirement that all the pictures should be taken in a sequence but it highly improves the speed of creating a model.

Computing a point cloud from 500 input images took more than 20 minutes to complete on our test machine. Nevertheless, the model creation step can be run on a server with multiple CPU and GPU cores to greatly improve the speed of reconstruction. Since the reconstruction time is increasing linearly, even larger models (from more photos) can be built in a reasonable time.

7.2.2 Locating user in a model

User localisation is the most important feature of an indoor positioning system. The performance of the system highly depends on the accuracy of

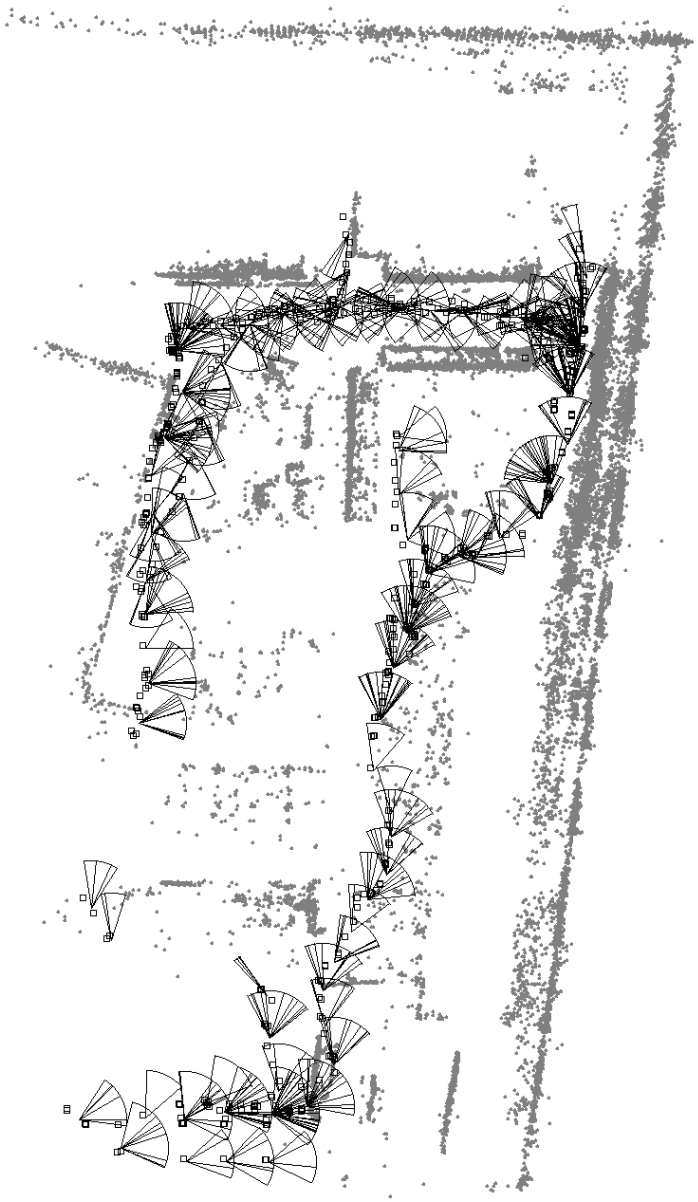

Figure 7.1: View angles of the photos and camera positions inside a library model

localisation, rates of false negatives, when the system is unable to localise a position by the supplied input, and the speed of processing localisation requests. In this section we assess our prototype system in terms of the mentioned aspects and explain the findings.

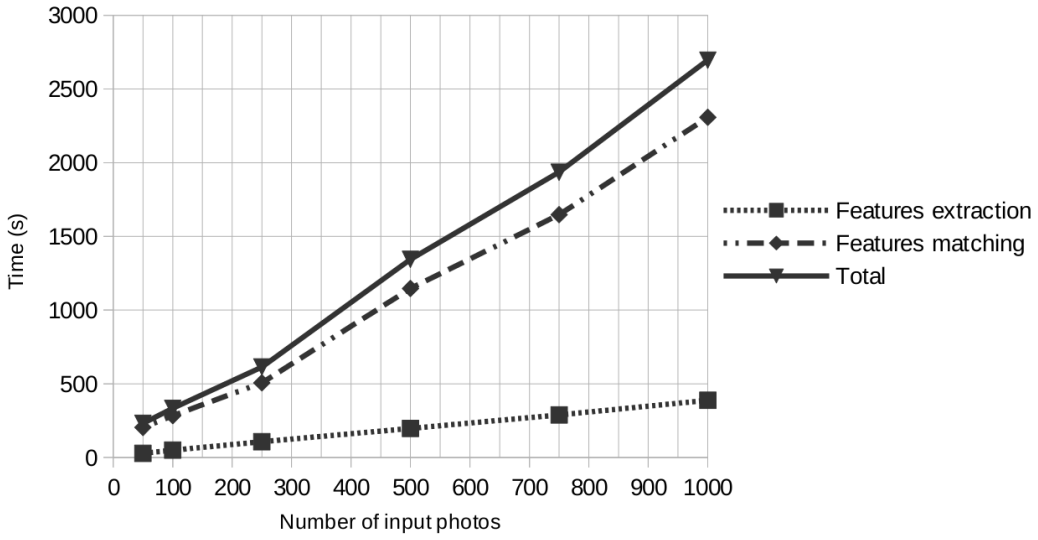

Figure 7.2: Time to create a 3D model from different number of photos

7.2.2.1 Accuracy assessment

The most important feature of a navigation system is the accuracy of positioning. It is important to know whether the system can locate a position of a user with an error as small as a few meters. In order to evaluate the accuracy, we used the library model built from 500 images. The library model was geo-referenced fixing the point (0, 0, 0) (zero point) near the entrance of the library. We used a measuring tape with an accuracy of 1.0 mm to measure distances in the library. Even though the accuracy of the measuring tape is high, the measurements are not accurate within millimetres. Since we were measuring distances manually and photos in the reference points were taken while holding a smartphone, there may be errors of several centimetres due to human factor. Therefore, the accuracy measurements here are expressed within centimetres.

For the test, we randomly chose 5 spots in the library and measured straight distances to the zero point. We also used another known reference point (2.06, 0, 0) to measure distance to 2 more spots. In each of the points we took 10 photos that were facing different angles and located their positions using our system. Since not all the photos were located successfully (the success factor of locating a photo is described in the following sections), we used first 5 located photos for the each spot. The positions of the located photos are show in Figure 7.3 a). The figure shows that some of the photos were located much better than the others. Especially the photos that were

facing dense areas of the model.

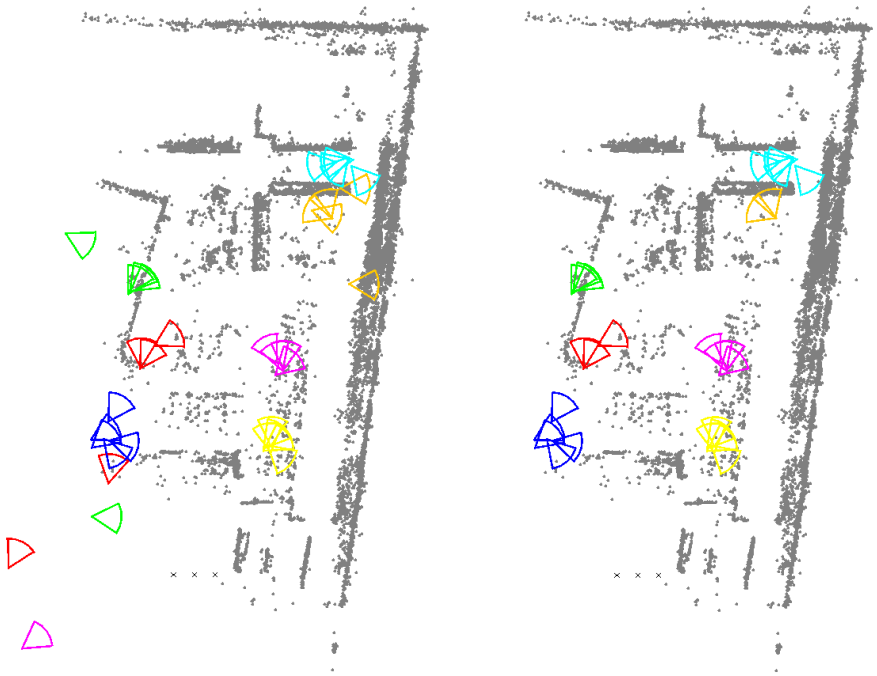

Figure 7.3: Positions of located photos. Locations of photos that were taken in the same spot are shown in the same colour. On the left: a) positions after successfully localising first 5 photos at the 7 spots; On the right: b) filtered results regarding camera focal length parameter

We then calculated distances from each photo to the reference point and compared them to the real world measured distances. We also observed the estimated focal length of each camera. The error rates of locating a photo and the corresponding focal length of the camera are shown in Figure 7.4. When a focal length is between 1300 and 1550 px, the error of locating is around one meter. However, for focal lengths that are beyond this range the error rates can go up to 9 meters. Therefore, we implemented a simple filtering algorithm that discards photos with incorrectly estimated focal lengths (see Figure 7.3 b)). After filtering, the average localisation error for the remaining photos was 0.26 m. The maximum error in the tested photoset was only 1.24 m.

As the results show, the largest error is due to invalid estimation of the photo focal length. The focal length estimation algorithm used by VisualSFM is close sourced, thus it is not possible to tweak the algorithm directly. Nevertheless, it is possible to filter photos with wrong focal lengths and thus highly

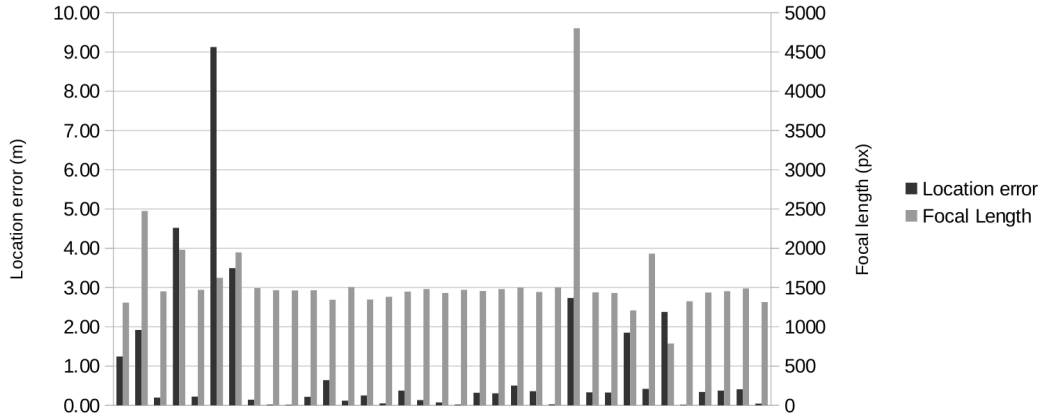

Figure 7.4: Error rates of locating a photo and the corresponding focal lengths of the cameras

improve the average accuracy of localisation. The filtering technique works with most of the mobile device cameras, since it is based on the camera focal length and the size of the optics inside various mobile devices are similar.

There are several more factors that contributed to the location errors. Firstly, the geo-referencing process imposes an error since we manually measured the distances between the cameras positions used for geo-referencing. Secondly, geo-referencing tool estimated a transformation error of 0.035 meters after geo-referencing the model.

7.2.2.2 Success ratio assessment

In worst cases, some photos may not be located at all. In order to evaluate the ratio of successfully locating a position of a user by the supplied photo, we took 100 random pictures in the area covered by the library model. Random photos also helped to identify what are the requirements and conditions for a photo to be located.

The model used for obtaining the rate of successful localisation was built from 500 photos. The top view of the model and the camera positions / orientations are shown in Figure 7.1. The pictures were taken in a random fashion, some of them facing similarly as the photos that were used for building the model, the others facing novel angles. None of the photos from the set were unfocused and blurry (knowing that the probability of locating such a photo is very low even in a highly dense SfM model).

Each of the 100 photos was supplied for locating in the model. The test

Figure 7.5: Successfully located images and their viewing directions (marked as patterned shapes)

reviled that 44 images were located successfully and others were not. The positions and orientations of located photos are shown in Figure 7.5. As seen from the figure, all the located photos are facing areas that contain many points and have an angle similar to pictures that were used to create the model.

An interesting question to answer is why the other photos were not localised. After inspecting the unlocalised photos we found a few factors that caused a failure in localising.

Firstly, some parts of our model contain a very sparse amount or no points at all (see Figure 7.5 bottom left corner). Obviously, the photos that were facing this way were not located.

Secondly, photos of small objects (e.g. only a chair, table or a small piece of a shelve is visible in a photo) were not successfully located (see Figure 7.6 a)). They did not contain enough overlapping features with the model.

Thirdly, the pictures with wrong focus points could not be located (see Figure 7.6 b)). The focus point of the photo was towards furniture that was moved but not on the shelves or walls.

Moreover, there were pictures taken at novel angles (see Figure 7.6 c)

Figure 7.6: Examples of not localised photos. From left to right: a) the photo contains too small objects; b) the photo is focused onto furniture but not on walls or shelves; c) the photo is taken at a very different angle

Figure 7.7: Effect of different lighting conditions for locating photos. From left to right: a) successfully localised photo with too much brightness; b) not localised photo with too much brightness

Figure 7.8: Not localised photos due to many changes in an area. From left to right: a) Photo of an area that was used to build the model; b) New photo supplied for localising (unsuccessful)

). Even though the photos cover some of the dense areas of the model, the angles were too large e.g. facing the completely opposite way than the cameras that were used in building the model.

Finally, different lighting conditions influenced some of the photos. The SIFT feature extractor is sensitive to lighting changes to a certain level.

Figure 7.7 shows that some photos with different lighting can be successfully localised (a), while others cannot (b).

Another factor that contributed to a low ratio of successfully located photos was that some pieces of furniture were moved between a time the model was built and the photos for locating were taken. Even some new pieces of furniture were placed in the library. The system can tolerate a small degree of changes. However, the photos that contain mostly the repositioned or new furniture were not located. (see Figure 7.8).

There are several solutions for improving the number of successfully localised photos in a model. We need to make sure, that a model is built from sufficient amount of photos and points densely cover all the areas. Also, photos for creating the model must be taken from many different angles. Finally, there must be a way to update the model when the surrounding area changes. Thus, the model represents the most recent view of the area and new photos can be localised. Of course, for a user it is always possible to take a blurry photo or a photo with a highly different lighting. Nevertheless, assuming that the ratio of such photos is low and regarding the photos from the test photoset and previously described solutions we could achieve a successful localisation ratio of more than 90%.

7.2.2.3 Localisation speed assessment

The last important feature of a navigation system is the ability to locate a position of a user in a reasonable time. In order to measure how long does it take to locate a user we utilised models built from 50, 100, 150, 250, 350 and 500 photos. All the models represent the same area - Aalto CS library. After that we randomly took 100 photos in the same area. We used those photos to estimate average time required to locate a user in models of different size.

The times required to locate pictures in different models are presented in Figure 7.9. As seen in the graph on the right, for a model that contains 500 photos, it takes around 70 seconds on average to process a location query. Successful queries are processed faster, while unsuccessful ones require more processing time. The chart on the left has an additionally plotted standard deviation of the results. Error bars represent one standard deviation to both positive and negative directions.

However, as seen from the chart, the standard deviation of the results is high. It is due to the nature of location query processing algorithm (described in section 4.4). It utilises heuristic techniques to skip some parts of the matching process and can dramatically increase the time needed to locate an image. Table 7.3 shows the minimum and maximum time spent to process a query in a model built form 500 images. Even though, unsuc-

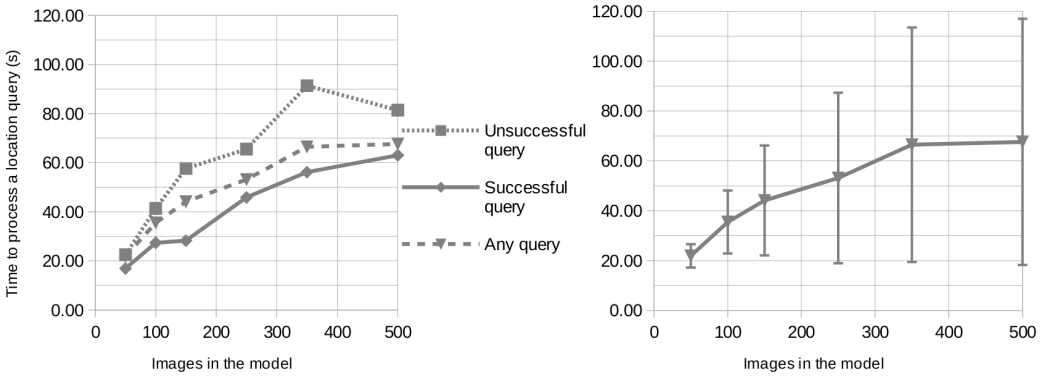

Figure 7.9: On the left: average times required to locate user's position in models built from different number of photos. On the right: average time to process location query and its standard deviation

cessful queries took more time to complete on average, the successful ones have higher extremes and the timing difference can be as large as 16 times. The tests were run on a single GPU system. Regarding that the most time is spent on feature matching and that the matching process can be easily parallelized, using a cluster of GPUs would dramatically decrease the time needed to process a location query.

Query	Minimum time (s)	Maximum time (s)
Successful	14.19	224.47
Unsuccessful	19.04	196.25

Table 7.3: Extreme values of processing a location query in the 500 photo model

7.3 Navigating

After obtaining the current location of the user and getting his or her destination, the navigation system has to find a path between the locations in a map. In order to be able to find a path, it firstly needs to build the area map from the previously created model. We evaluated the time needed to extract obstacles information from the point cloud, and to build the navigation mesh and find a path in the built mesh.

For the experiments we utilised models built from 50, 150, 200 and 400 photos. For each of the model we run the shape extraction and navigation

map building algorithm. The results of the tests are presented in table 7.4. "Time to build" refers to the time needed to extract the shapes and save the navigation map as an *.OBJ* file. "Extracted" shapes shows how many independent obstacles were found in the point cloud and "Extracted points" tells how many points were used to represent the 2D shapes of the obstacles. The shape extraction operation executes fast in compare with the time needed to build a point cloud from the same amounts of photos. Since shape extraction has to be executed only after the model is changed (e.g. added new photos), it is not an often executed algorithm and does not impose a performance penalty to the system.

Images in a model	Time to build (s)	Extracted shapes	Extracted points
50	1.051	33	637
150	0.644	45	890
200	1.153	39	1005
400	5.108	50	1138

Table 7.4: Evaluation of extracting obstacles information from the model

A more time critical part is finding a path in the map. This type of request is executed each time the position of the user changes. The evaluation of the time, needed to find a path in models of different size, is presented in Table 7.5. "Time to build" refers to the time needed for Recast library to build a navigation mesh out of the supplied mesh; "Navigation time" shows the time needed for Detour library to find a path and for our system to process the result and prepare a response with the extracted path. Since the operation takes a very short time it is expressed in milliseconds (ms) instead of seconds (s). It turns out, that navigation task is a fast operation. It can be optimised even more by removing the time component for building the navigation mesh, since the same navigation mesh can be reused to serve many path requests in the same area.

Images in a model	Time to build (s)	Navigation time (ms)	Total time (s)
50	0.027	0.062	0.027
150	0.037	23.62	0.061
200	0.038	19.22	0.057
400	0.041	37.24	0.079

Table 7.5: Evaluation of path finding speed

Chapter 8

Discussion

In this section, we discuss the main results and contributions of the thesis. We discuss the assumptions and the current state of the implementation, the aspects of its evaluation, and finally, the lessons learnt during the thesis and ideas for the future work.

The thesis work defines, implements and evaluates a prototype of an infrastructure less indoor navigation system. The developed navigation system relies only on computer vision techniques, in particular - Structure form Motion (SfM) - to build a map for navigation, and finds the shortest paths using an A* (A star) algorithm. The system consists of a server and a client. The server is responsible for building the navigation map and handling navigation queries, while the client supplies photo data for localisation and displays navigation instructions to a user. After a client supplies a photo of an area to the server, the server locates the current position of a user. After the user chooses a destination, the server plans a shortest path between the current location and the destination and sends the results back to the client application. The user can see detailed visual instructions which help to find a way through the area. The created system prototype was successfully tested in a 300 m^2 indoors area.

8.1 System implementation

The current implementation follows client – server architecture. For the prototype system, the server application programming interface (API) exposes only the functions related to navigation. Model creation, growing and processing to obtain a navigable mesh is done locally on the server side. For this reason, a GUI tool was created that allows building a model from photos, geo-reference it, extract obstacles and build the navigation mesh. The

tool even allows trying out the navigation by supplying photos to locate a start and a destination. More details about the created tool are presented in chapter 6.

The server exposes REST based API over HTTP. It is sufficient to use plain HTTP since the requests/responses are large in size in compare with a HTTP header. Thus, the protocol overhead is negligible. There is no need to keep an alive TCP connection between the client and the server, since the communication is stateless. For our system, the rate of requests can be optimised on the client side and decrease its energy consumption. In contrast, radio or GPS based navigation systems require the radio receiver interface to be enabled all the time during the navigation process.

The server utilises Restful (stateless) design, thus it can serve multiple clients without memory storage overhead. The model (map) used to process the queries is the same for all queries. VisualSFM, the tool that we use to perform SfM operations required locating a user, works by accessing and writing some of the model files, thus multiple access would corrupt the file state. Before deploying the system for a wide use, we need to apply one of the solutions to control multiple resource access. One way is to copy the model files for each request. Copying the whole model may impose an unwanted processing overhead. However, it is possible to copy only the files that may be changed during SfM operations and only link the other resources. Another way is to programmatically control the access to the shared resources while processing multiple queries. The latter solution gives more flexibility in control and avoids any file copying overhead.

8.2 Model creation

Our positioning algorithm is based on SfM models represented as point clouds. Thus, in order to be able to obtain a position in an area, a 3D model of the area has to be built. During the research we tried several different approaches and techniques for building a model. We found out, that the quality of input photos has the largest impact on the success of model creation. While SfM algorithm accepts any photo as a valid input, the quality of a produced point cloud highly varies.

First of all, we need to define what factors describe the quality of the model. The first important factor is whether a produced model is continuous or fragmented. SfM produces fragmented models if there is not enough information (photos) about certain spots in the area. The fragmented model is not useful for our system since the small models are not aligned and have different coordinate systems. Secondly, an important metric to consider is an

amount of points in a model. As our evaluation tests revealed, the amount of points in a model is proportional to the success rate of locating a photo in the model. Also, the points may not be distributed evenly through the area. Some places of the model can be densely covered with points while others - sparsely. Thus, the sparse areas would have a lower rate of successful localisation than the dense ones.

While building a model, our target is to build a continuous and densely covered by points model with an optimal amount of photos. Clearly, if we have a vast amount of photos the possibility of building a good quality model is higher but the time needed to process all the photos may be unacceptably long. Also, our model creation algorithm uses sequence matching instead of full matching, which also imposes a constraint on a photos collection technique. The constraint means, that once we start taking photos, each pair of neighbouring images should have enough overlapping area. Thus, the angle between photos should be no more than 15 degrees. Also, an object that is visible in photos should be present in at least 3 consecutive pictures to be reconstructed. And, of course, to fully recreate an object form all 4 directions we need at least 24 images to be taken around it. Since we are taking photos of indoor areas, mostly rooms, a strategy of taking photos should be carefully planned to obtain the best results. A good approach is to start from the entrance and walk around the room in a clockwise direction, taking several photos at each step. Once the entrance is reached again, the same room must be traversed counter-clockwise in order to increase the chances of locating any new photos in the room. Of course, while making any turns, the line of photos should not be broken and at each 15 degrees of camera rotation a new photo should be taken. Regarding larger rooms and areas, it is better to take photos in smaller sets e.g. 50 photos per set. After that, models can be built from the photosets and merged together. An important requirement is that the sets should have enough overlapping information to be successfully merged.

The final factor that highly impacts the reconstructed model quality is the area where the photos are taken. Even though many areas are suitable for our system, we found it really difficult to reconstruct monotonic and low textured areas such as long corridors or rooms containing only plain coloured walls. It is due to SIFT feature extraction algorithm which cannot find enough features in photos that contain little texture information. The photos that were taken for tests in the Aalto CS library contained several thousands of features while the photos taken in a plain corridor had only several hundreds of extracted features. Another obstacle in rebuilding models is glass and reflective surfaces. Glass cannot be detected and the glass walls may eventually be marked as a traversable area in the navigation mesh. What

is more, reflective areas may create obstacles in the places which are actually empty and distort the model. A solution to overcome some of the problems is to utilise a different feature extraction algorithm such as ASIFT or SURF, or even a combination of the algorithms. So far none of the SIFT based feature extraction algorithms can deal with glass and reflective surfaces. However, if there are any stickers or other objects attached to the glass walls or windows, those parts will be detected and we can infer that the way is blocked and the navigation path will be planned around the glass object.

8.3 Localisation accuracy

Navigation system evaluation shows the accuracy of localising a position of a user. Currently, we can achieve 0.26 metres accuracy for successfully localised photos. As noted in Chapter 7, the system is not able to localise all the input photos if they are taken in a random fashion. Nevertheless, regarding the analysis of the unsuccessfully located photos, with a very few improvements the system would be able to localise up to 90% of the input images.

There are several ways to improve the overall accuracy of the system. Firstly, the navigation map building process itself has to be robust. The quality of a point cloud influences the overall quality of the system. Some improvements for 3D model building part were described in a previous section. However, not only model building but also geo-referencing step influences the accuracy. If the model is improperly referenced, locating errors are unavoidable. To make the referencing more robust, several places of the point cloud should have referenced points. For example, all the entrances to an area could have referenced camera positions. In this way, if there are referenced points all over the area, errors would not accumulate in places that are further from the referenced areas.

Secondly, some of the SfM parameters can be tuned for processing location queries. For example, the number of features found in a query image could be increased. Again, a different feature extraction algorithm could be used.

Finally, several images could be taken at the same location point. After locating all of them, the average position could be used. We could also perform an analysis of the photo to discard blurry ones or the ones with incorrectly estimated camera parameters. Initially we discard photos that have an incorrectly estimated focal length. However, we need to develop a more robust algorithm for determining the ranges of correct focal lengths for different cameras. So far the algorithm is suitable for photos taken by smartphone cameras. In this way the average position of located photos will not be affected by outliers.

Additionally, the obstacle information from the model could be utilised to fix positions that were located on top of obstacles. The system could even be combined with other indoor positioning techniques, such as dead-reckoning to help estimating the position.

8.4 Environmental and social impact

Global Positioning System (GPS) helped to develop a huge variety of useful applications ranging from social smartphone apps to industrial applications and emergency systems. Yet, there is no comparable solution for indoors navigation. The work done in the thesis may lead to a navigation system that does not require infrastructure and can be used with regular consumer devices. With a few improvements it can be used to enhance navigation in obscured citizen areas such as metro stations, government offices, museums or huge shopping malls. An important part is that our system does not require infrastructure. Thus, it can be utilised in areas where it is impossible or too expensive to install any additional infrastructure.

With integration with GPS, such a navigation system could provide seamless navigation experience wherever a user goes. The system would not only give a better understanding of the indoor areas for visitors, it could highly benefit visually impaired persons to find a way through. The system can even be used to guide unmanned vehicles / robots indoors. Applications for unmanned, image based indoor navigation guided vehicles could include anything from controlling a forklift in a warehouse to controlling a wheelchair in a hospital.

While having lots of potential application scenarios, the system may raise privacy issues. The localisation and navigation queries are processed on a remote server. Since a client supplies data obtained from his or her current position, the system has a theoretical ability to track an individual in an area. However, the current implementation of the system uses stateless design. It means that no state is saved on the server and all location and navigation queries are one time requests, thus disabling the possibility to track individuals.

Of course, the system should be deployed on a large scale server infrastructure, since it is a resource demanding system. Intensive computations consume large amounts of energy. Fortunately, data centres are optimised for efficiency and the performance/energy consumption ratio is constantly growing. Additionally, it is an infrastructure-less system and only requires a client device to run a front end application. Since the client application can run on a consumer mobile device, the users of the system need no additional

hardware to use the system. Even if a client is not a smart device user (e.g. robot), the only requirements for the system are a cheap image sensor and a network connection.

8.5 Future work

The prototype navigation system which was implemented during the thesis is a first step towards large scale infrastructure-less and accurate indoors navigation system. There are many ways to improve the system and make it more robust and user friendly.

8.5.1 Crowd sourced approach of data collection

Current navigation system works in a 300 m^2 area. Yet it was cumbersome for one person to collect all the photos for building a model. A way to improve the speed and scale of model creation step is to implement a crowd sourced photo collection approach.

Currently, the model building process was executed solely on the server, after manually storing the photos on the same machine. With a very small effort, the server could accept photos from a mobile client. A mobile client could send a small set of photos that could be instantly added to the model. With such function enabled, multiple mobile clients could contribute by supplying photos.

The challenging part would be to persuade users to take photos of an area and contribute to the model creation. Some approaches could be to implement a game-like photo collection where a user is rewarded points for taking photos in particular area. The photos contributed and the part of the model that was created by those photos could be tagged by the photographer's name (or nickname). Ideally, photos should be taken according to the rules discussed in "Model creation" paragraph. There should be a mobile client application that would guide the user, by giving instant hints on screen about how to take pictures and of which area. The approach could be similar to the one used in applications which allow constructing panorama or 360 degrees view images from photos. The camera view could be enriched with augmented reality details to better guide the user. Additionally, the model construction server should give instant feedback whether and how the newly uploaded photos contributed to the existing model.

Updating the information of the model would be another useful feature. Indoor environments tend to change: cabinets or tables are moved, new shops open in a shopping mall, market is redecorated during the sales season and

so on. Therefore, if a user supplies an photoset that look different from the current photos that were used to build a model of a particular area, the system should update the area model by using the new photos and removing the old ones. A challenging part is not only to remove the relevant points from the point cloud model but also to find the correct position of the new photoset and add the new points.

8.5.2 Large scale real world scenarios

Our 3D model that we utilised for most of the tests was created from 500 photos. In order to cover areas as big as universities or shopping malls, tens of thousands of photos may be needed. Even if we can use crowd sourcing to collect the photos, there is still a challenge to process such an amount of information. So far our system can manage and grow a model by adding smaller already built models to it. However, it was not benchmarked with thousands of photos. There may arise issues other than the speed of reconstruction, for example storage and memory constraints (a model built from 500 photos requires more than 700MB storage space). There needs to be a way to make an area model modular. For instance, each floor of the building could be stored as a different model. Then, there must also be a database of stairs, elevators and any other means of going from one model to another to make the navigation process smooth.

8.5.3 Own Structure from Motion implementation

So far, all SfM operations are performed by VisualSFM tool. Since only some parts of the tool are open sourced, it is hard to adapt the tool to work for the navigation system needs. The models generated by VisualSFM are stored on disk. Then, a model has to be read to the memory for further processing by our own program. Since SfM operations are not performed directly in the same process, exchanging data between VisualSFM and our application processes causes additional overhead (reading / writing to the disk). VisualSFM has many parameters to tune and has a wide CLI functionality but not everything can be configured and due to the use of pseudo-random numbers in the code[1] it does not guarantee the same output for an identical input set. Therefore, it would be valuable to have a fully open sourced, customisable SfM pipeline that would perform as fast as VisualSFM but would allow a higher level of customisation and would prevent data exchange overhead.

[1] https://groups.google.com/forum/#!topic/vsfm/BfDRCSiBBww

8.5.4 Navigation process

Currently, a user has to find her or his way through the area by using the instructions provided from the server. The instructions provide an overview map, as well as photos of the area along the way. Nevertheless, it can be further improved to make the navigation process even easier.

One way would be to use more photos in the instructions or even provide similar results as in Google Street View application[2]. However, such approach would require an initial model built from a very large amount of images. Another approach would be to utilise augmented reality elements on top of the camera view. The application could draw a navigation path as an augmented reality object with some minimal textual information at the turning points.

Constant checking of the current position of a user would also be beneficial for navigation purposes. Such feature is mandatory in order to correctly implement the augmented reality navigation algorithm, since it cannot rely only on smartphone accelerometer and gyroscope sensors for more than several meters. Therefore, an application should automatically take pictures of an area and relocate the current position. Since localisation queries are resource demanding, for a system optimisation, the query could be performed e.g. every 30 seconds or every minute, assuming an agent moves no faster than a regular walking speed.

8.5.5 Parallel computing at backend

Drawbacks of computer vision algorithms are high complexity and a relatively long processing time. Even though Structure from Motion operations are resource demanding, the algorithms can be easily executed in parallel. It would dramatically increase the speed of building the model and would decrease the latency for processing location queries.

One way to implement concurrent SfM pipeline is to employ open source distributed computing frameworks such as Apache Hadoop[3] or Apache Spark[4]. The Hadoop framework is based on map-reduce paradigm and easily scales on clusters containing thousands of machines. The framework was proven useful for managing big data and is used by hundreds of IT companies[5]. The Spark framework is a fast and general solution for a large scale data processing. It is said to be up to 10 times faster than the default Hadoop

[2]https://www.google.com/maps/views/streetview
[3]http://hadoop.apache.org/
[4]http://spark.apache.org/
[5]http://wiki.apache.org/hadoop/PoweredBy

implementations and if running only in-memory can be faster as many as 100 times[6]. The Hadoop framework can be used for building SfM models, since the model creation step can be executed when the whole system load is low (e.g. at night). The Spark framework is more light weight and can be used to enhance the speed of processing real time queries by running the feature matching and bundle adjustment steps on a cluster of machines.

Another way is to use not a CPU based cluster but GPU based one, since feature extraction, matching and some parts of bundle adjustment can be executed solely on GPUs, resulting in even faster execution.

[6]http://spark.apache.org/

Chapter 9

Conclusions

The major contributions of the Thesis work are (1) implementation of a functional infrastructure-less computer vision based indoor navigation system, (2) analysis and integration of current open and free to use Structure from Motion (SfM) and pathfinding tools, (3) analysis of performance issues of creating SfM models and proposed ways of speeding up the process, (4) evaluation of the created image based indoor navigation system and review of feasibility of such a system.

The main aim of the Thesis was to develop an indoor navigation system that would work based on computer vision techniques. We defined the functional and non-functional requirements for our system and analysed the current indoor navigation methods. We developed an architecture for our system and after that, we implemented our own solution that meets the requirements. Finally, we evaluated the performance of the system in terms of localisation accuracy and processing speed.

To achieve high accuracy and an infrastructure-less implementation of the navigation system, we used SfM point cloud building techniques and navigation mesh based path-finding algorithms. The map for locating a user is represented as a 3D point cloud and the same point cloud is used to extract obstacles and construct a navigation mesh of the area. The point cloud is built from plain 2D photos of the area. Later on, a user is localised by simply supplying a new photo taken by her or his smartphone in a current area. After a user selects a destination, the path is planned by the navigation algorithm in an already generated navigation mesh and visual instructions on how to navigate in the area are presented to the user.

Although we used open sourced and free to use software, we had and managed to solve many issues and technical challenges to adapt the software to our needs and to overcome current limitations of the software. Our optimised, sequence matching based algorithm for creating 3D models allows

creating point cloud models in a reasonable time. We also developed a model merging algorithm to enable an incremental creation of SfM models. We implemented our own solution for extracting obstacles information from a point cloud and to construct a mesh suitable for navigation.

The software, created during the Thesis work, includes a graphical tool for creating and growing point cloud models, extracting obstacles information and generating a navigation mesh, analysing point clouds and visualising localisation results. We developed a server application for handling localisation and navigation queries and a mobile client application which is used as a front-end by the user of the navigation system.

The experimental approach of solving the research problem proved successful, since not only a working prototype was implemented but we were also able to evaluate the system in a real world scenario. The system was proved to be able to localise a position within an average error of 0.26 meters and a maximum error of 1.24 m. We also presented the ways to improve the overall success ratio of localisation by improving the quality of the 3D point cloud and by filtering input photos. Even though navigation map building is still a time and resource demanding operation and the time needed to localise a photo is long, by using a cluster of high-end machines with multiple CPU and GPU cores we could adapt our system to serve localisation queries in real time.

Even though, SfM model creation may not be always robust and cannot guarantee to work in any environment, our system was successfully tested inside Aalto Computer Science building library and shows very promising results. The same system should demonstrate similar performance in areas such as shopping malls or train stations.

Lastly, the knowledge, gathered during the Thesis, and the various issues that had to be solved in order to build this system forms a strong background for the future work on infrastructure-less image based navigation systems.

Bibliography

[1] Fixing pathfinding once and for all. http://www.ai-blog.net/archives/000152.html, July 2008. [Online]. Accessed: 2014-04-15.

[2] Abdel-Hakim, A. E., and Farag, A. A. Csift: A sift descriptor with color invariant characteristics. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (2006), vol. 2, IEEE, pp. 1978–1983.

[3] Arikawa, M., Konomi, S., and Ohnishi, K. Navitime: Supporting pedestrian navigation in the real world. *Pervasive Computing, IEEE 6*, 3 (2007), 21–29.

[4] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[5] Becker, S., Peter, M., Fritsch, D., Philipp, D., Baier, P., and Dibak, C. Combined grammar for the modeling of building interiors. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci* (2013).

[6] Bestmann, A., and Reimann, R. Easypoint – indoor localization and navigation low cost, reliable and accurate.

[7] Borovikov, I. Navigation graph generation. http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/navigation-graph-generation-r2805, June 2011. [Online]. Accessed: 2014-06-01.

[8] Champandard, A. Are waypoint graphs outnumbered? not in alienswarm! http://aigamedev.com/open/reviews/alienswarm-node-graph/, July 2010. [Online]. Accessed: 2014-04-26.

[9] Codreanu, V., Dong, F., Liu, B., Roerdink, J. B., Williams, D., Yang, P., and Yasar, B. Gpu-asift: A fast fully affine-invariant feature extraction algorithm. In *High Performance Computing and Simulation (HPCS), 2013 International Conference on* (2013), IEEE, pp. 474–481.

[10] Demyen, D., and Buro, M. Efficient triangulation-based pathfinding. In *AAAI* (2006), vol. 6, pp. 942–947.

[11] Duckham, M., Kulik, L., Worboys, M., and Galton, A. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition 41*, 10 (2008), 3224 – 3236.

[12] Edelsbrunner, H., Kirkpatrick, D., and Seidel, R. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on 29*, 4 (1983), 551–559.

[13] Fallah, N., Apostolopoulos, I., Bekris, K., and Folmer, E. Indoor human navigation systems: A survey. *Interacting with Computers 25*, 1 (2013), 21–33.

[14] Fischer, C., Muthukrishnan, K., Hazas, M., and Gellersen, H. Ultrasound-aided pedestrian dead reckoning for indoor navigation. In *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments* (2008), ACM, pp. 31–36.

[15] Furukawa, Y., Curless, B., Seitz, S., and Szeliski, R. Towards internet-scale multi-view stereo. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (June 2010), pp. 1434–1441.

[16] Irschara, A., Zach, C., Frahm, J.-M., and Bischof, H. From structure-from-motion point clouds to fast location recognition. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 2599–2606.

[17] Kazhdan, M., Bolitho, M., and Hoppe, H. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (2006).

[18] Ke, Y., and Sukthankar, R. Pca-sift: A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* (2004), vol. 2, IEEE, pp. II–506.

[19] Kim, J., and Jun, H. Vision-based location positioning using augmented reality for indoor navigation. *Consumer Electronics, IEEE Transactions on 54*, 3 (2008), 954–962.

[20] Li, Y., Snavely, N., Huttenlocher, D., and Fua, P. Worldwide pose estimation using 3d point clouds. In *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., vol. 7572 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 15–29.

[21] Lodha, S. K., and Xiao, Y. Gsift: geometric scale invariant feature transform for terrain data. In *Electronic Imaging 2006* (2006), International Society for Optics and Photonics, pp. 60660L–60660L.

[22] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[23] Morel, J.-M., and Yu, G. Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences 2*, 2 (2009), 438–469.

[24] Moulon, P., Monasse, P., Marlet, R., et al. Global fusion of relative motions for robust, accurate and scalable structure from motion. In *Proceedings of IEEE International Conference on Computer Vision* (2013).

[25] Park, S., and Hashimoto, S. Autonomous mobile robot navigation using passive rfid in indoor environment. *Industrial Electronics, IEEE Transactions on 56*, 7 (2009), 2366–2373.

[26] Sattler, T., Leibe, B., and Kobbelt, L. Fast image-based localization using direct 2d-to-3d matching. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (2011), IEEE, pp. 667–674.

[27] Sinha, S. N., Steedly, D., and Szeliski, R. A multi-stage linear approach to structure from motion. In *Trends and Topics in Computer Vision*. Springer, 2012, pp. 267–281.

[28] Skiena, S. S. *The Algorithm Design Manual*, 2nd ed. Springer Publishing Company, Incorporated, 2008.

[29] Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: exploring photo collections in 3d. *ACM transactions on graphics (TOG) 25*, 3 (2006), 835–846.

[30] Szeliski, R. *Computer vision: algorithms and applications*. Springer, 2010.

[31] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. Bundle adjustment — a modern synthesis. In *Vision Algorithms: Theory and Practice*, B. Triggs, A. Zisserman, and R. Szeliski, Eds., vol. 1883 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pp. 298–372.

[32] Wang, A., Qiu, T., and Shao, L. A simple method of radial distortion correction with centre of distortion estimation. *Journal of Mathematical Imaging and Vision 35*, 3 (2009), 165–172.

[33] Wu, C. Siftgpu: A gpu implementation of scale invariant feature transform (sift). `http://cs.unc.edu/~ccwu/siftgpu/`, 2007. [Online]. Accessed: 2014-06-12.

[34] Wu, C. Towards linear-time incremental structure from motion. In *3DTV-Conference, 2013 International Conference on* (2013), IEEE, pp. 127–134.

[35] Wu, C., Agarwal, S., Curless, B., and Seitz, S. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* (June 2011), pp. 3057–3064.

[36] Wu, C., Agarwal, S., Curless, B., and Seitz, S. M. Schematic surface reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (2012), IEEE, pp. 1498–1505.

[37] Wu, J., Cui, Z., Sheng, V. S., Zhao, P., Su, D., and Gong, S. A comparative study of sift and its variants. *Measurement Science Review 13*, 3 (2013), 122–131.