Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago de Matos Ferreira Madeira

Melhoria do Alinhamento de Imagens RGB-D Usando Marcadores Fiduciais

Enhancement of RGB-D Image Alignment Using Fiducial Markers

Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago de Matos Ferreira Madeira

Melhoria do Alinhamento de Imagens RGB-D Usando Marcadores Fiduciais

Enhancement of RGB-D Image Alignment Using Fiducial Markers

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Miguel de Jesus Dias, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Miguel Armando Riem de Oliveira, Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor João Paulo Costeira
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores do Instituto Superior Técnico, Universidade de Lisboa (arguente)

Professor Doutor Paulo Miguel de Jesus Dias
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos /
acknowledgements

Gostaria de agradecer aos meus orientadores por me guiarem, pela partilha de conhecimento, pela paciência incansável e pelas valiosas contribuições para o trabalho. À minha família e amigos mais próximos, pelo incentivo, atenção e carinho. Aos meus colegas e companheiros, pelo apoio e camaradagem.

Palavras Chave

Visão por Computador, Optimização Geométrica, Calibração de Câmaras, Reconstrução 3D, Inpainting, Marcadores Fiduciais, Nuvens de Pontos, Projeção de Pontos 3D

Resumo

A reconstrução 3D é a criação de modelos tridimensionais a partir da forma e aparência capturadas de objetos reais. É um campo que teve origem em diversos ramos da visão computacional e computação gráfica, e que ganhou grande importância em áreas como a arquitetura, robótica, condução autónoma, medicina e arqueologia. A maioria das tecnologias de aquisição de modelos atuais são baseadas em LiDAR, câmeras RGB-D e abordagens baseadas em imagens, como o SLAM visual. Apesar das melhorias que foram alcançadas, os métodos que dependem de instrumentos profissionais e da sua operação resultam em elevados custos, tanto de capital, como logísticos. Nesta dissertação foi desenvolvido um processo de otimização capaz de melhorar as reconstruções 3D criadas usando uma câmera RGB-D portátil, disponível ao nível do consumidor, de fácil manipulação e que tem uma interface familiar para o utilizador de smartphones, através da utilização de marcadores fiduciais colocados no ambiente. Além disso, uma ferramenta foi desenvolvida para permitir a remoção dos ditos marcadores fiduciais da textura da cena, como um complemento para mitigar uma desvantagem da abordagem adotada, mas que pode ser útil em outros contextos.

Keywords

Computer Vision, Geometric Optimization, Camera Calibration, 3D Reconstruction, Inpainting, Fiducial Markers, Point Clouds, Projection of 3D points

Abstract

3D reconstruction is the creation of three-dimensional models from the captured shape and appearance of real objects. It is a field that has its roots in several areas within computer vision and graphics, and has gained high importance in others, such as architecture, robotics, autonomous driving, medicine, and archaeology. Most of the current model acquisition technologies are based on LiDAR, RGB-D cameras, and image-based approaches such as visual SLAM. Despite the improvements that have been achieved, methods that rely on professional instruments and operation result in high costs, both capital and logistical. In this dissertation, we develop an optimization procedure capable of enhancing the 3D reconstructions created using a consumer level RGB-D hand-held camera, a product that is widely available, easily handled, with a familiar interface to the average smartphone user, through the utilisation of fiducial markers placed in the environment. Additionally, a tool was developed to allow the removal of said fiducial markers from the texture of the scene, as a complement to mitigate a downside of the approach taken, but that may prove useful in other contexts.

Contents

Contents i

List of Figures iii

Glossary vii

1 Introduction 1
1.1 Scope 1
1.2 Motivation 2
1.3 Objective 2
1.4 Document structure 3

2 State of the Art 5
2.1 RGB-D cameras 5
2.2 Registration 5
2.3 SLAM 8
2.4 Bundle Adjustment 8
2.5 Structure from Motion 9
2.6 Proposed approach 10

3 Experimental infrastructure 13
3.1 The device: ZenFone AR (Google Tango Enabled) 13
3.2 Software 15

4 Methodology for Optimization 19
4.1 Dataset management: OCDatasetLoader 19
4.2 Aruco information: OCArucoDetector 20
4.3 Generic Optimization Module: OptimizationUtils 20
4.4 Optimization case study: Optimization of Camera Poses 24

4.4.1 To output a dataset . . . 29

5 Methodology for Fiducial Marker Removal . . . 31
5.1 Creation of the masks . . . 32
5.2 Navier-Stokes-based Inpainting . . . 34
5.3 Improving the inpainting . . . 35
5.3.1 First stage . . . 35
5.3.2 Second stage . . . 36
5.3.3 Final Results . . . 38
5.4 Inpainting non-detected markers: Cross-Inpainting . . . 39
5.5 3D colour visualisation . . . 41

6 Results . . . 45
6.1 Evaluation methodology . . . 45
6.2 Collecting datasets . . . 45
6.3 FARO dataset . . . 46
6.4 Evaluation procedure . . . 47
6.5 MeetingRoom Dataset . . . 48
6.5.1 Optimization of Camera Poses Results . . . 48
6.5.2 Fiducial Marker Removal Results . . . 51
6.6 Lobby Dataset . . . 52
6.6.1 Optimization of Camera Poses Results . . . 52
6.6.2 Fiducial Marker Removal Results . . . 54

7 Conclusions . . . 55

References . . . 57

List of Figures

2.1 Registration of 3D point clouds to create a complete 3D model. 6
2.2 Flowchart of the registration process. 6
2.3 Representation of feature detection and matching, using feature descriptor. 7
2.4 Simple example of triangulation. 9
2.5 Creating a 3D reconstruction using SfM. 10
2.6 Representation of detection and matching of aruco markers for a pair of captures. 11

3.1 ASUS ZenFone AR sensors setup. 14
3.2 Main features of Google Tango. 14
3.3 Cloud Compare software screenshot. 17
3.4 Meshlab software screenshot. 17

4.1 Flowchart of the Generic Optimization implementation. 21
4.2 Fragment of a sparse matrix. 27
4.3 Default visualisation function of OptimizationUtils: Graph of the value of error associated with each residual. Initial reprojection error in green, current reprojection error in blue. . 28
4.4 Example of reprojection visualisation for one of the images of the dataset. Aruco marker detection in red (target of optimization), initial projection in green, current projection in dark blue. 28
4.5 3D representation of the position of the cameras and aruco markers in the scene, produced by the visualisation function. 29
4.6 Matplotlib's qualitative colormap tab10. 29
4.7 Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 1). Before optimization on the left, after optimization on the right. 30
4.8 Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 2). Before optimization on the left, after optimization on the right. 30
4.9 Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 3). Before optimization on the left, after optimization on the right. 30

5.1 Simple mask created from fiducial marker detection. 32
5.2 Mask after dilation was applied. 33
5.3 Mask created from the projection of the 3D object. 33
5.4 Results for 2D dilation and 3D projection methods for mask creation. 33
5.5 Mask used for marker removal. 34
5.6 Navier-Stokes-based inpainting from `OpenCV` *inpaint*. 34
5.7 Mask for the pixels which will be replaced by the ones in Figure 5.8. 35
5.8 Auxiliar image 1. Median blur of Figure 5.6. 35
5.9 First stage result. 36
5.10 Mask for the pixels which will be replaced by the ones in Figure 5.11. 36
5.11 Auxiliar image 2. Median blur of the first stage result. 37
5.12 Second stage result. 37
5.13 Original image from OfficeDemo dataset. 38
5.14 Restored image from OfficeDemo dataset. 38
5.15 Illustration of conversion between different reference frames. 39
5.16 Final mask before optimization. Masks obtained from aruco marker detection in red and masks obtained from projections from other cameras (Cross-Inpainting) in blue. 40
5.17 Final mask after optimization. Masks obtained from aruco marker detection in red and masks obtained from projections from other cameras (Cross-Inpainting) in blue. 40
5.18 Restoration result before optimization. 41
5.19 Restoration result after optimization. 41
5.20 Point cloud coloured with texture obtained from restored images, using cross-inpainting. Before optimization on the left and after optimization on the right (meshlab). 42
5.21 Point cloud coloured with texture obtained from restored images, no cross-inpainting. Before optimization on the left and after optimization on the right (meshlab). 42
5.22 Merged point cloud coloured with texture obtained from images. Before optimization on the left and after optimization on the right (meshlab). 43
5.23 Merged point cloud coloured with texture obtained from images (detail). Before optimization on the left and after optimization on the right (meshlab). 43

6.1 FARO generated point cloud after downsampling. 46
6.2 FARO generated point cloud after downsampling and clipping. 47
6.3 MeetingRoom dataset coloured with texture obtained from original images. Before optimization on the top and after optimization on the bottom. 48
6.4 MeetingRoom dataset coloured with texture obtained from original images (detail 1). Before optimization on the top and after optimization on the bottom. 49
6.5 MeetingRoom dataset coloured with texture obtained from original images (detail 2). Before optimization on the top and after optimization on the bottom. 49

6.6 MeetingRoom dataset colormapped with Hausdorff distance to FARO dataset. Before optimization on the top and after optimization on the bottom. 50
6.7 MeetingRoom dataset colormapped with Hausdorff distance to FARO dataset (detail). Before optimization on the left and after optimization on the right. 51
6.8 MeetingRoom dataset coloured with texture obtained from original images on the top and texture obtained from restored images on the bottom. 51
6.9 Lobby dataset coloured with texture obtained from original images. Before optimization on the top and after optimization on the bottom. 52
6.10 Lobby dataset coloured with texture obtained from original images (detail). Before optimization on the left and after optimization on the right. 53
6.11 Lobby dataset detail coloured with Matplotlib's tab10 colormap. Before optimization on the left and after optimization on the right. 53
6.12 Lobby dataset coloured with texture obtained from original images on the top, and texture obtained from restored images on the bottom. 54

Glossary

ICP	Iterative Closest Points
AR	Augmented Reality
DEM	Departamento de Engenharia Mecânica
UA	Universidade de Aveiro
XML	Extensible Markup Language
LiDAR	light detection and ranging
SLAM	Simultaneous Localisation and Mapping
IR	Infrared radiation
ADF	Area Description File
ToF	Time-of-Flight
IMU	Inertial Measurement Unit
SfM	Structure from Motion
UAS	Unmanned Aerial Systems
LM	Levenberg-Marquardt
LBS	location-based services
FLANN	Fast Library for Approximate Nearest Neighbours
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features
HOG	Histogram of Oriented Gradients
VIO	Visual-Inertial Odometry

CHAPTER 1

INTRODUCTION

This chapter describes the context of the problem, the motivation behind the work carried out, along with its objectives, ending with an overview of the document's structure.

1.1 SCOPE

3D reconstruction is the creation of three-dimensional models from the captured shape and appearance of real objects. It is a field that has its roots in several areas within computer vision and graphics, and has gained high importance in others, such as architecture, robotics, autonomous driving, medicine, and archaeology. Most of the current model acquisition technologies are based on light detection and ranging (LiDAR) [1], [2], RGB-D cameras [3], [4], and image-based approaches, such as visual Simultaneous Localisation and Mapping (SLAM) [5]. Despite the improvements that have been achieved, methods that rely on professional instruments and operation result in high costs, both capital and logistical [6].

The introduction of low-cost RGB-D cameras, which can be used as hand-held devices, created an opportunity for 3D reconstruction of scenes to be performed at consumer-level. These instruments allow for the creation of textured 3D models, while being easy to operate. Considering the high probability of object occlusion in indoor environments, meaning an object is hidden (occluded) by another object, fixed scanners, which usually require wide space and several poses, are less practical for indoor reconstruction tasks. The manoeuvrability of hand-held devices, allowing the user to get closer to parts of the scene and capturing them from different angles, proves to be an advantage. As such, even though they are generally less accurate than fixed scanners such as LiDAR, mobile or hand-held scanners are known to be more suitable to perform indoor scanning [7]. These devices are also compelling for applications such as indoor navigation, since the way they are operated inherently conveys information about the empty space, which often corresponds to the navigable space, as the device is carried through it, by a person or a robot for instance.

1.2 MOTIVATION

Indoor 3D models have great potential in object tracking and interaction, scene understanding, virtual environment rendering, indoor localisation and route planning, amongst others [8]–[10]. Given the rapid development of location-based services (LBS) and indoor applications, fast acquisition and high-fidelity reconstruction of complete indoor 3D scenes has become an important task [11]. Currently there is extensive research done into live 3D reconstruction techniques, where the final model is built during the capturing of the data. However, post-processing techniques show significant potential and need for future research, particularly in 3D reconstruction using RGB-D cameras [12]. As such, it would be interesting to complement the existing live 3D reconstruction techniques that use low-cost RGB-D hand-held cameras, considering their advantages and placement at consumer-level in the market, with some automatic post processing, to see whether their results can be improved, creating significant additional value.

1.3 OBJECTIVE

The main objective of this dissertation is to develop an optimization procedure capable of enhancing the 3D reconstructions created using a hand-held RGB-D camera, through the utilisation of fiducial markers placed in the environment.

In this project, a lot of importance was given to the idea of implementing something that could be reused for future work. Approaching the problem by creating a base implementation of standard tools, with useful abstractions, and defining a structure that would take advantage of these tools for future implementation of optimizations. Facilitating future work, making development time shorter, and allowing more resources to be spent on key elements of the optimization. We ourselves already had a few ideas for future utilisation of these tools, other projects that require an optimization to be performed, such as the calibration of a set of sensor in an autonomous vehicle and colour consistency correction in 3D reconstructions. Whilst researching geometric optimization, experimentation with different techniques, different cost functions, and different approaches requiring additional information, the idea is to allow the specialisation of the researcher and the focus of their work to be this, and not everything else which must be put in place around it. We propose the development of an api for optimization problems, which allows a more intuitive and systematic approach for testing various algorithms, and then the utilisation of these tools for a case study, where the previously discussed optimization procedure is implemented.

1.4 DOCUMENT STRUCTURE

This dissertation is composed of seven chapters, arranged as follows:

Introduction The current chapter, in which the problem is placed into context, the motivation behind the work carried out is discussed, along with its objectives.

State of the Art Presents a general overview of the registration problem in the Computer Vision context and how it relates to SLAM, Structure from Motion (SfM) and Bundle Adjustment. The hardware of interest, RGB-D devices, is also introduced. The chapter culminates with the explanation of how our solution fits into this outlining.

Experimental Infrastructure Briefly describes the software and hardware utilised to develop this work.

Methodology for Optimization Describes the methodology used to implement the tools and their main functionalities.

Methodology for Fiducial Marker Removal: Describes the methodology used to remove the fiducial markers from the texture of the scene, as a way to mitigate the effects of the technique used on the final results.

Results Showcases the experimental results obtained and presents some commentary about them.

Conclusions Contains conclusions about the work developed and possible future work.

CHAPTER 2

State of the Art

This chapter contains some detail about the hardware on which the work was focused, RGB-D cameras, and the main subjects this dissertation is related to, starting in a more high-level abstraction, with the registration problem and SLAM, then presenting some important algorithms, namely Bundle Adjustment and Structure from Motion, and ending with the explanation of how our solution fits into this scenario.

2.1 RGB-D CAMERAS

RGB-D cameras are capable of obtaining the depth of an object and the corresponding texture information, through the combination of a depth image and a standard RGB image, respectively. The depth image is usually obtained by a depth sensor utilising a Time-of-Flight (ToF) measurement system [13] or Structured Light triangulation [14], [15]. ToF works by measuring the round trip time of an artificial light signal, in this case Infrared radiation (IR), to resolve the distance between the camera and the target object. Structured light works by projecting known patterns on to the scene and then measuring the way they are deformed, allowing for the calculation of depth and surface information. Modern RGB-D approaches are mostly based on the fundamental research by Curless and Levoy [16] who introduced the work of volumetric fusion, providing the foundation for the first real-time RGB-D reconstruction methods [17].

2.2 REGISTRATION

Registration is the process of alignment of multiple captures, with different viewpoints, of the same scene or object. The result may be an extended version of a 2D image, such as a panoramic photograph, or a 3D representation of the scene. Within the context of 3D reconstruction, registration translates to the problem of aligning the multiple point clouds

that make up the 3D model of the scene, see Figure 2.1. When using RGB-D cameras, in which the RGB data is registered with respect to the depth data, it is also possible to align the RGB data instead. In this case, it is often a better idea to do so, as opposed to registering the point clouds directly, because depth data is usually less accurate than RGB data.

Consider the example of an indoor reconstruction performed with a hand held device. This device would capture information as it travels through multiple positions in the environment, allowing for the collection of data from various places of the scene in different angles, places that may be occluded, or too far to be captured from the initial viewpoint of the acquisition. In order to obtain the completed 3D model, these multiple parts of the scene must be matched together. The alignment of point clouds is a problem directly tackled by Iterative Closest Points (ICP) [18]. However, this algorithm works by minimising the difference in a pair of point clouds. In order to align multiple ones, the algorithm must be applied many times. If this is done in a chain, for example, it creates a very real possibility of a drift, a cumulative error that grows as each pair is fit together.

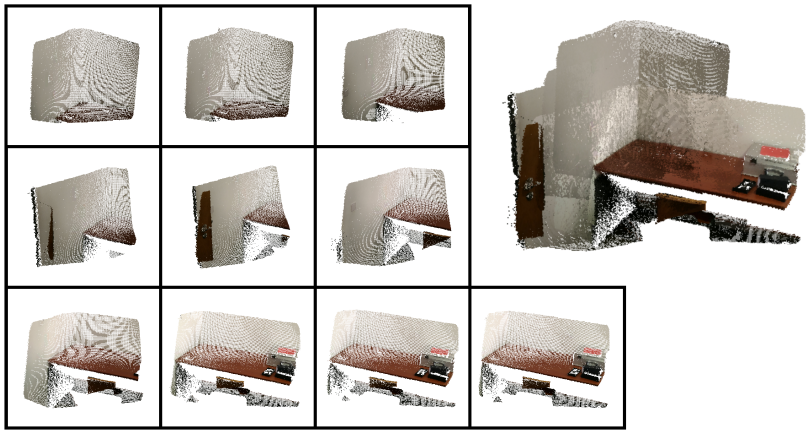

Figure 2.1: Registration of 3D point clouds to create a complete 3D model.

In most cases, the registration process consists of performing the same general steps for every pair of captures. In each pair, only one of the captures will be transformed using the estimated model and the other will remain the same. The latter one is referred to as reference capture. An outline of these steps is represented in Figure 2.2, followed by a brief description.

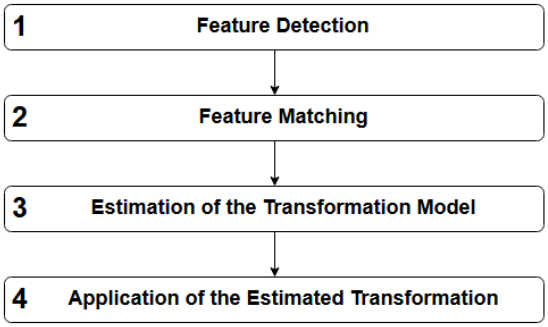

Figure 2.2: Flowchart of the registration process.

1 Feature Detection

Distinctive features of the captures, such as blobs, edges, contours, line intersections or corners are detected in order to find correspondences between them. These features are represented in a feature vector, as represented in Figure 2.3[1], in such a way that transformations like scaling and rotation will not influence the feature identifier [19], examples of this would be Scale-Invariant Feature Transform (SIFT)[20], Speeded Up Robust Features (SURF)[21], and Histogram of Oriented Gradients (HOG)[22].

The most common algorithms used in feature detection are Canny and Sobel for edge detection, Harris and SUSAN for edge and corner detection, Shi-Tomasi and Level curve curvature for corner detection, FAST, Laplacian of Gaussian, and Difference of Gaussians for corner and blob detection, MSER, PCBR, and Gray-level blobs for blob detection [23].

2 Feature Matching

After the features have been identified in both captures of the pair, they are searched for matches, i.e. features that are identical in both captures. There is a vast range of different approaches to solve this problem, from brute-force matcher and Fast Library for Approximate Nearest Neighbours (FLANN) to pattern recognition [24], all of them very dependent on the type of scene and the available processing time. This step is still a challenge and there is always the possibility that a significant number of detections will be matched incorrectly.

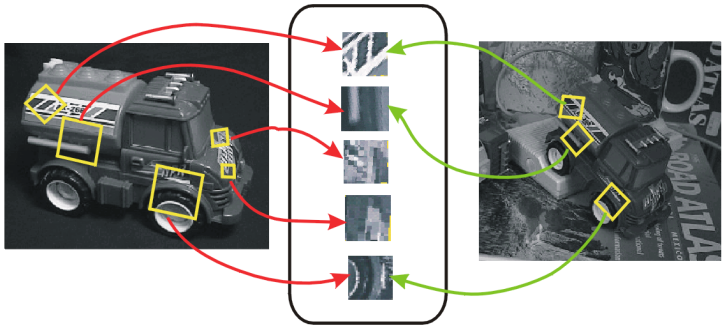

Figure 2.3: Representation of feature detection and matching, using feature descriptor.

3 Estimation of the Transformation Model

As mentioned before, the feature matching process seldom works perfectly. Matching features incorrectly means it is impossible to find a transformation model that works for all matches found. One way to solve this problem is by recursively estimating a model that works for a subset of matches, until the size of that subset is considered large enough, RANSAC [25]. The remaining matches are discarded. Another approach is using ICP [18], this algorithm does not require an individual feature matching process in some implementations [26]. While there is still a matching procedure, it is merely based on distance. Each feature is paired with its closest neighbour of the reference capture and the transformation model is recursively

[1] In `https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf`

estimated through a hill climbing algorithm, so the distance between neighbours approaches zero.

4 Application of the Estimated Transformation
The capture is transformed by application of the estimated transformation model.

2.3 SLAM

Simultaneous Localisation and Mapping (SLAM) [27] refers to the problem of trying to localise some sensor in the environment, while simultaneously mapping the structure of that environment. This can be done in many different ways, with different sensors.

In the context of this dissertation, Visual SLAM is of the most interest. It utilises 3D vision to perform location and mapping, becoming an optimisation problem where the goal is to compute the best configuration of camera poses and point positions, in order to minimise the average reprojection error. That is, the difference between a point's detected location in an image and where it is expected to be, given the camera pose estimate. The method of choice to solve this problem is called bundle adjustment, a nonlinear least squares algorithm which, given a suitable starting configuration, iteratively approaches the minimum error for the whole system, as explained in section 2.4.

2.4 BUNDLE ADJUSTMENT

Given a set of measured image feature locations and correspondences, the goal of bundle adjustment is to find 3D point positions and camera parameters that minimise the reprojection error. This optimization problem is usually formulated as a non-linear least squares problem, where the error is the squared ℓ^2 norm, or Euclidean norm, of the difference between the observed feature location and the projection of the corresponding 3D point on the image plane of the camera (reprojection error). The Levenberg-Marquardt (LM) algorithm [28] is the most popular algorithm for solving non-linear least squares problems, and the algorithm of choice for bundle adjustment.

In 2010, Agarwal et. al. [29] presented the design and implementation of a new inexact Newton type Bundle Adjustment algorithm. Consider the existence of a series of 3D points in the real world. These points are captured in images by different cameras, each camera being defined by its orientation and translation relative to a reference frame, its focal length and distortion parameters. After the desired acquisitions are completed, the 3D points are projected into the images and the 2D coordinates are then compared to the ones obtained by feature detection in the images. The goal being to adjust the initial estimation of the camera

parameters and the position of the points, in order to minimise the reprojection errors, i.e.

$$\min_{\hat{\mathrm{P}}^i, \widehat{\mathbf{X}}_j} \sum_{ij} \ell^2 \left(\hat{\mathrm{P}}^i \widehat{\mathbf{X}}_j, \mathbf{x}_j^i \right)^2 \tag{2.1}$$

where ℓ^2 is the Euclidean norm, $\mathbf{x}_j^i$ are the coordinates of the j-th point as seen by the i-th camera, $\hat{\mathrm{P}}^i$ is the projection matrix of the i-th camera, and $\widehat{\mathbf{X}}_j$ are the 3D points [30].

2.5 STRUCTURE FROM MOTION

Structure from Motion (SfM) is a technique that utilises a series 2D images to reconstruct the 3D structure of a scene or object. SfM can, using several images captured by one or multiple cameras, produce point cloud based 3D models, similar to those obtained by RGB-D cameras or LiDARs. This technique can be used to create models of objects with consumer-grade digital cameras. This technique has been made possible by advances in computers, digital cameras, and Unmanned Aerial Systems (UAS). Together, these advances have made it feasible for a wide range of users to be able to generate 3D models, without extensive expertise or expensive equipment. It works based on the same principles as stereoscopic photogrammetry. In stereophogrammetry, triangulation is used to calculate the relative 3D positions (X, Y, Z) of objects from stereo pairs. A simple example can be seen in Figure 2.4, where common points B and O are identified within each image and a line of sight can be constructed from the camera location to the points on the object. The intersection of these lines determines the three-dimensional location of the points.

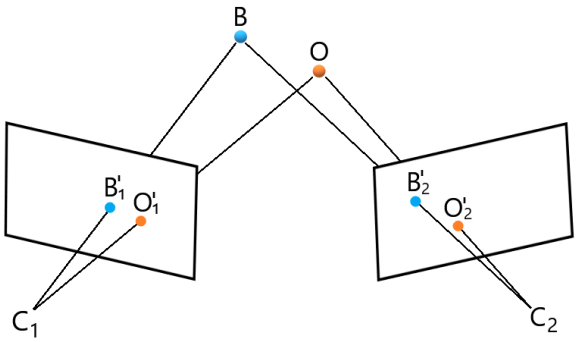

Figure 2.4: Simple example of triangulation.

Through SfM, it is possible to create a 3D reconstruction of an area or an object, using a set of images with a high degree of overlap, taken from different angles, as demonstrated in Figure 2.5[2]. The camera does not need to be specialised, standard consumer-grade cameras work well for SfM methods. The images are often captured from a moving sensor, but can also be taken by a person or multiple people, with different cameras, in different locations and angles.

[2]In http://theia-sfm.org/sfm.html

Some specialised software packages are able to automatically identify matching features in multiple images. These distinctive features are often corners or line segments, see section 2.2. In the following step, instead of trying to align captures, as in Figure 2.2, the matching of the features is used to produce estimates of the camera positions and orientations (pose) and the 3D coordinates of these said features, producing a point cloud. It is important to note for section 2.6 that, the better these estimates are, the better alignment of the captures we should expect to see. A key component of this process in most SfM systems is Bundle adjustment, defined as the joint non-linear refinement of camera and point parameters. This solves a problem analogous to visual SLAM, see section 2.3, the main difference being that SLAM is usually meant to work in real-time on an ordered sequence of images, while SfM approaches often work on an unordered set of images as a kind of post-processing, many times done in the cloud.

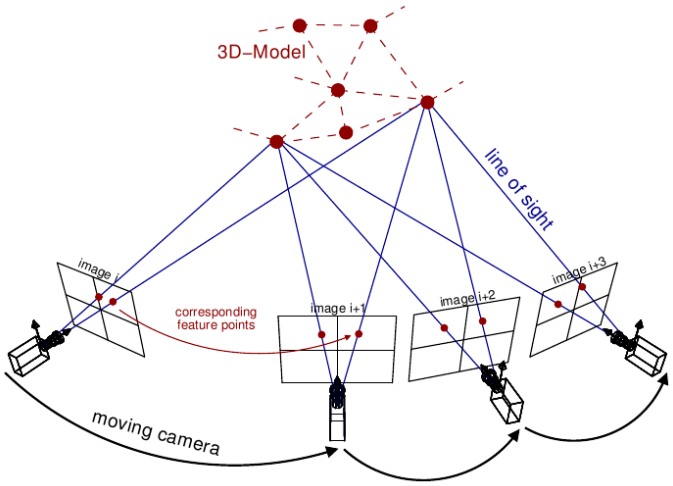

Figure 2.5: Creating a 3D reconstruction using SfM.

2.6 PROPOSED APPROACH

As mentioned in section 1.3, the main objective of this dissertation is to develop an optimization procedure capable of enhancing the geometry of 3D reconstructions created using a consumer-level hand-held RGB-D camera, utilising fiducial markers placed in the environment, by applying an automatic post processing that targets the alignment of RGB-D data, making use of Bundle Adjustment.

One problem to deal with, when tackling the registration problem, as discussed in section 2.2, is that the matching of detected features between captures is not very reliable. In the case of Bundle Adjustment, the minimisation of the reprojection may be computed using a wrongly matched feature in a pair of images, causing errors in the alignment of captures. Therefore, we have chosen to make use of fiducial markers, particularly aruco markers [31], [32], to ease the feature detection and matching step, placing them on the environment as

the visual features to be detected for this post-processing. Aruco markers are binary square fiducial markers that can be used for camera pose estimation. Their main benefit is that their detection is robust and fast. Each marker has an identifier (id), recognisable by the processing of a small grid of black and white pixels. The detection and matching of features will depend on the processing and identification of this id, see Figure 2.6, making it very hard for false matching to occur. Although under poor lighting conditions or in blurred images some aruco markers may go undetected, it is highly improbable that they will be identified with the wrong id.

Figure 2.6: Representation of detection and matching of aruco markers for a pair of captures.

The biggest downside to this approach is the pollution of the acquired scene's texture with these fiducial markers. Therefore, along with the optimization procedure, an additional tool was developed, that allows the automatic removal of aruco markers from the texture of the scene, taking advantage of the relationship between different captures.

CHAPTER 3

Experimental infrastructure

This chapter briefly describes the hardware and software used in this project. The device used to create the initial 3D reconstruction, Zenfone AR, is showcased in section 3.1 and all the software used is described in section 3.2.

3.1 the device: zenfone ar (google tango enabled)

The platform chosen to obtain the starting point for the developed solution was Google Tango. The reasoning behind this choice is that it is a consumer-level product that is widely available, accessible, easily handled, and with a familiar interface to the average smartphone user. The 3D reconstruction to be enhanced could have been produced by another platform and device. As far as our implementation goes, this choice influences only the format of the available data and the original structure of the datasets in disk.

Google Tango is a project developed by Google Inc. that aimed to create a mobile device capable of 3D reconstruction, powered by computer vision, image processing, and special sensors. The set of sensors includes the components of an RGB-D camera, along with a motion tracking camera, and was embedded into a few ordinary, consumer-level Android devices. The one being explored in this dissertation is the `Zenfone AR`. Along with an RGB camera and an IR depth sensor (which make up the RGB-D camera), and the motion tracking camera, see Figure 3.1[1], this device also takes advantage of an accelerometer, gyroscope, ambient light sensor, barometer, compass, and GPS, commonly found in smartphones.

Tango enabled devices are capable of capturing and processing information about their surrounding environment. For this purpose, three main technologies are used: motion tracking, area learning and depth perception, see Figure 3.2[2].

[1] In `https://www.techpinas.com/2017/01/asus-zenfone-ar-specs-features.html`
[2] In `https://www.asus.com/pt/Phone/ZenFone-AR-ZS571KL/`

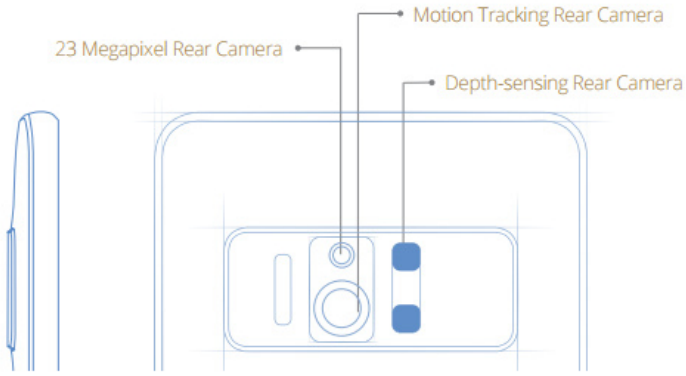

Figure 3.1: ASUS ZenFone AR sensors setup.

Figure 3.2: Main features of Google Tango.

Motion tracking technology allows the device to know, in real time, its own position and orientation (pose) in the 3D space. For Google Tango, a Visual-Inertial Odometry (VIO) [33] approach was adopted. It is based on feature tracking in images, combined with the analysis of the data provided by the Inertial Measurement Unit (IMU) [34] of the device. Images are captured with the motion tracking camera that benefit of a large field of view (up to 160°). Image processing algorithms are used to detect features, such as corners and edges, and study their optical flow across the acquired frames (up to 60 images/s), which, in this context, may be defined as the pattern of apparent motion of the features between consecutive captures, caused by the movement of the camera. In addition, the accelerometer and the gyroscope (forming the IMU) provide orientation and acceleration of the device. This combined information is used to obtain the pose of the device at all times.

Area learning in Tango works based on the well known Simultaneous Localisation and Mapping (SLAM) feature-based approach [27]. This approach extracts sparse features from the sensors and creates a map exclusively comprised of them. In Tango, these features are saved in an Area Description File (ADF) during the motion tracking process. Such information provides some advantages, namely making the motion tracking more accurate by performing correction of drifts, which occur due to the cumulative error and loss of accuracy.

Depth perception relies on an IR depth sensor. This sensor, as described in section 2.1,

relies on a ToF technique to measure the distance of the device to the surrounding objects. It can handle a range of objects at a distance from 0.5m to 4m, and, much like IR sensors in general, does not perform well while scanning areas lit with high IR light sources, or objects that do not reflect IR light, such as black surfaces.

3.2 SOFTWARE

This section consists of brief descriptions of the pieces of software used in this dissertation. It is divided in Smartphone apps, Libraries and Frameworks, File formats, and Graphical Software.

Smartphone apps

Open Constructor Android app from Google Play that enables Tango compatible devices to perform 3D reconstructions. A modified version, which allows the saving of the internal dataset, was used. This dataset is used as the starting point for the optimization implemented. The Open Constructor app essentially functions as an interface between the Google Tango api and our optimization api.

Libraries and Frameworks

Python is a general purpose programming language that became popular for its syntax and gentle learning curve. It is the de facto language for science, along with MATLAB. However, unlike MATLAB, it is open source. It benefits from a very large community and plenty of libraries that provide many algorithms and efficient data structures. It is also a dynamically-typed, garbage-collected language, which positively affects development time.

SciPy is an open source library, which contains routines commonly used in scientific work. There are routines for computing integrals numerically, solving differential equations, optimization, operations using sparsity matrices, etc. This library is implemented in the python programming language. One of those routines, useful for Large-scale bundle adjustment optimization problems, is utilised in this dissertation (*scipy.optimize.least_squares*).

Numpy is a library that contains an implementation of nd-arrays, as well as algorithms to manipulate them. This library was fundamental for this work to store and process data. The main advantage of this library is that it is implemented in compiled languages like C and Fortran, allowing for high performance and optimized data structures, all available through a clean interface in Python.

OpenCV is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. It contains a large number of optimized

algorithms, which include a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. Some examples are: object detection, recognising full or partial objects, camera calibration, and object tracking. OpenCV has C++, Python, and Java interfaces, being supported in Windows, Linux, Mac OS, iOS, and Android.

Module ArUco is an extra module of the OpenCV library. It contains functionalities to detect and identify aruco markers, as well as obtaining the marker pose relative to the camera reference frame in the 3D space.

File formats

PLY or Polygon File Format is one of the most used and supported file formats to store three dimensional data, like point clouds and meshes. It was originally developed and used in the Stanford University to store data from 3D scanners. It supports a wide number of properties, such as colour, transparency, surface normals, and texture coordinates.

JPEG is a commonly used format for images and it is used to store the RGB image of each capture in an Open Constructor dataset.

YAML is an human-readable format that is used to store parameters of the acquisitions, namely the calibration of the sensors. The advantage of this format is that files are very easy to read and modify by the user.

XML or Extensible Markup Language is a markup language that allows the definition of a set of rules for encoding documents in a format that is human-readable. It provides a flexible way to electronically share structured data via the Internet. As such, it was used to manage the shared dataset information, being processed by the dataset sharing script.

TXT is a plain text document. The Open Constructor app uses this format to save transformations, in separate files, relative to each capture in the dataset.

Graphical Software

CloudCompare is a software to render, process, and manipulate 3D point clouds. It includes many algorithms, namely point cloud registration, re-sampling, handling scalar fields, and automatic or interactive segmentation. It can render point clouds using different shaders and support point cloud decimation, which is a technique that allows manipulation of large point clouds without a decrease in performance. It was used to merge and downsample laser scans to be used as a groundtruth for the result analysis. CloudCompare was also used to apply the ICP algorithm to each point cloud and the laser scan result, so they would be transformed to the same frame of reference, and could then be compared. A screenshot of this software can be seen in Figure 3.3.

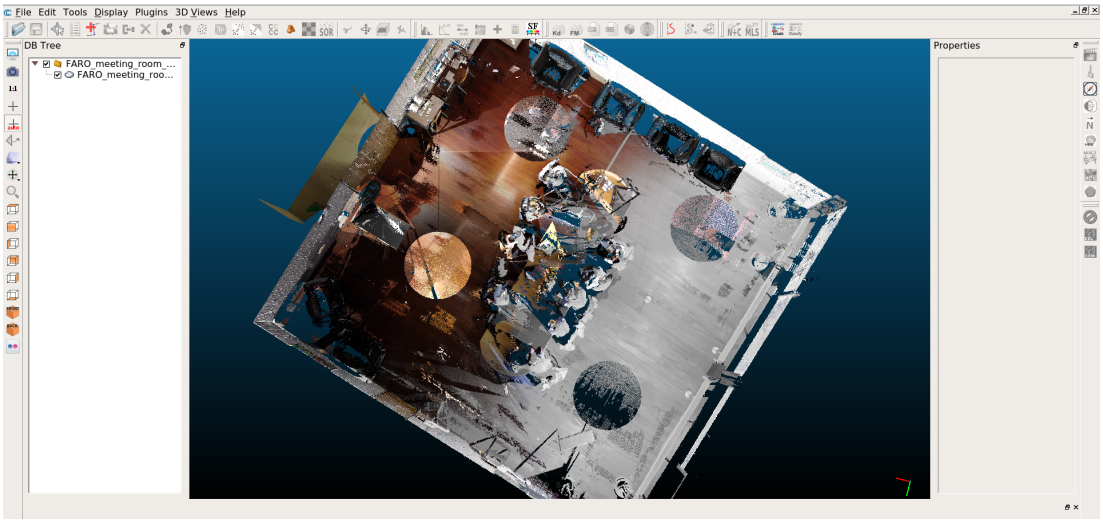

Figure 3.3: Cloud Compare software screenshot.

Meshlab is a 3D mesh processing software system that is oriented to the management and processing of unstructured large meshes and provides a set of tools for editing, cleaning, healing, inspecting, rendering, and converting these kinds of meshes. MeshLab is a free and open-source software. It was used to observe and compare coloured point clouds, to calculate Hausdorff distance between them, and to apply the values obtained as a colour coded quality function. A screenshot of this software can be seen in Figure 3.4.

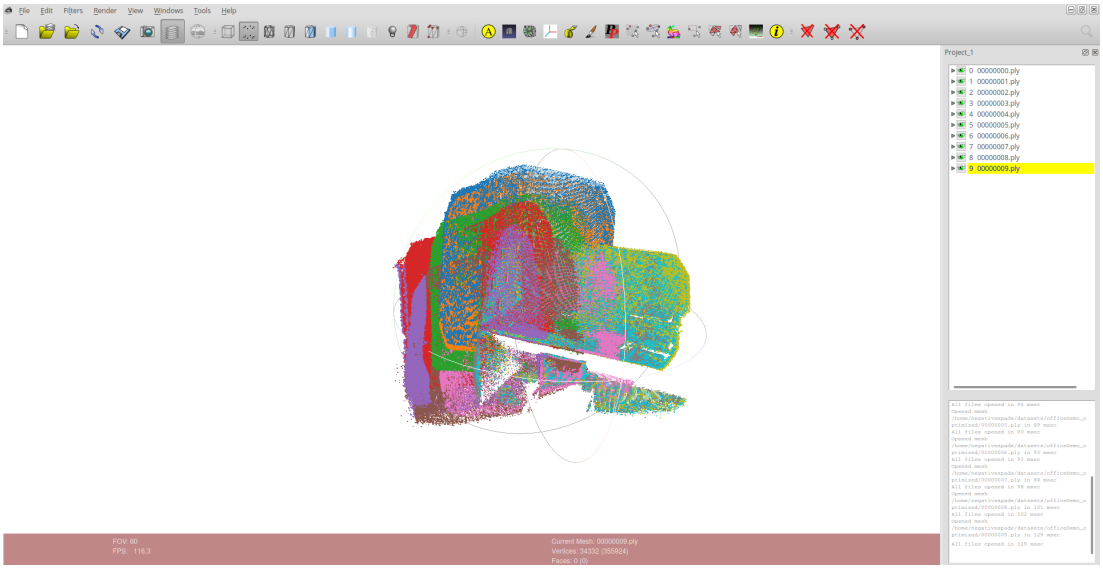

Figure 3.4: Meshlab software screenshot.

CHAPTER 4

Methodology for Optimization

This chapter describes the methodology for the implementation of a generic optimization procedure, followed by a case study where it is applied to the problem of enhancing the geometry of 3D reconstructions created using an RGB-D camera. The software developed was divided into the following modules:

OCDatasetLoader is a loader class for OpenConstructor datasets. It contains the essential functionalities to load all the information from disk to memory, keeping it in a well defined structure that makes its use logical and coherent.

OCArucoDetector implements detection and pose estimation of aruco markers fit for an OpenConstructor dataset, keeping information in a well structured manner. Also contains useful visualisation functions.

OptimizationUtils is a set of utilities and wrappers for using the python scipy optimizer functions.

4.1 Dataset management: OCDatasetLoader

The datasets used in this work were generated by the Open Constructor for Tango app. A tool called OCDatasetLoader (Open Constructor Dataset Loader) was created to manage all the information required from these datasets in disk, namely images, transformations, and point clouds, which are saved in `.png`, `.txt`, and `.ply` formats respectively. This information is kept in a well defined structure that makes its use practical.

This module also holds the code for dealing with the processing of all the required command line arguments. This includes the paths to various files, options for visualisation, and criteria for filtering of the information loaded.

The datasets are normally used by multiple people working in these projects. As such, a practical way of sharing them and easily keeping up with changes was developed. The datasets are compressed and uploaded to the cloud. A script was created to automatically download and extract each dataset to a local folder. There is also the option to obtain single datasets by name or URL. The list of available datasets is managed easily through an Extensible Markup Language (XML) file, containing the name, URL, and extension of each archive.

4.2 ARUCO INFORMATION: OCARUCODETECTOR

The OCArucoDetector (Open Constructor Aruco Detector) module was created to complement the data loaded by the OCDatasetLoader, in optimizations where information about aruco markers in the scene is required. This tool contains the utilities to detect and estimate the pose of the aruco markers in all the images of the dataset. It also implements abstractions of conversions between translation and Rodrigues form to transformation matrices, visualisation functions for detections, and the 3D visualisation of the aruco markers' pose in the scene.

4.3 GENERIC OPTIMIZATION MODULE: OPTIMIZATIONUTILS

This section describes a generic optimization implementation, making use of the tools implemented, and showcasing its main features. In Figure 4.1, an outline of the steps for said implementation is represented, followed by a brief explanation of each of them. In section 4.4, the concretization of these steps is detailed for the case of the optimization of camera poses.

Optimization functions usually receive all the parameters to be optimized within a single vector. In complex optimization problems, such as 3D reconstruction, the data structures involved tend to be naturally more complex than a vector. Hence, the need arises to use more complex data structures to ease the comprehension and aid challenging problem solving, by keeping the implementations more intuitive. On the other hand, the variables contained in these data structures must then be placed within a vector before they can be optimized. The approach taken to tackle this problem was to develop an api that uses the concept of *data model*. A data model is a generic data structure, which may be quite complex in some cases, and that holds in memory a set of parameters. As many data models as needed may be defined for one implementation. The is that these models help maintain the information structured and separated, and the code sane. When a new parameter is introduced to the optimization, one must indicate what data model it is saved in and two functions: the getter, a function that, given the data model as an argument, returns the current value of the parameter; the setter, given the data model and a value as arguments, this function updates the field within the data model that corresponds to the parameter. The definition of getters and setters for all the parameters we want to optimize allows for the full synchronisation between the

values within the vector that is optimized and the corresponding fields within the data models. Moreover, during the optimization, this synchronisation is done automatically by the api, every iteration, before the call of the cost function, keeping the whole program consistent. The main advantage of this is that the functions implemented by the user, namely cost and visualisation functions, need not retrieve the data from the vector, and may instead use the data models, which having been defined by the user themselves, should make things simpler and more intuitive.

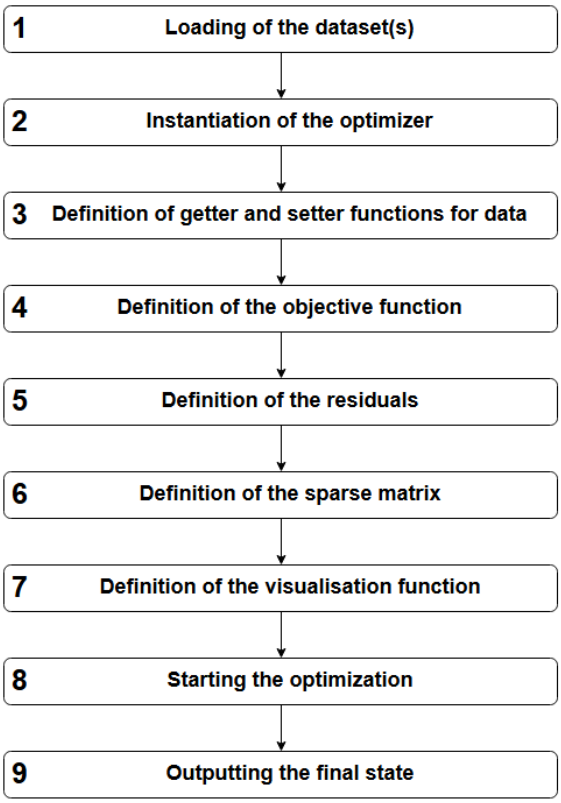

Figure 4.1: Flowchart of the Generic Optimization implementation.

1 Loading of the dataset(s)

The dataset loader is instantiated and used to load all the needed information to memory. This could be done using OCDatasetLoader, see section 4.1, another tool, or multiple tools.

2 Instantiation of the optimizer

This optimizer is a class implemented in the OptimizationUtils module. This module contains a set of utilities that try to simplify the workflow when using the scipy optimizer functions. The information loaded from the dataset can be structured and added to the optimizer as a data model, utilising the optimizer.*addModelData* function. A data model may be defined as any combination of data structures, such as dictionaries, lists, ndarrays, etc, because the interface with it (getter and setter) is defined by the user. At this stage, any other data models may be added to be used in the optimization.

3 Definition of getter and setter functions for data

Using the set of optimizer.*pushParam* functions, it is possible to define specialised getter and setter functions for each parameter of each data model. This is done to abstract the necessary conversions to copy information from the data models to the vector used in the optimization and vice versa. At this point, one should make sure the initial value, or first guess, for each parameter is stored correctly in the datamodel where the getter will retrieve it from.

4 Definition of the objective function

The objective function, or cost function, must be defined and set for the optimizer (optimizer.*(*setObjectiveFunction)). This is the function that computes the vector of errors associated with the optimization state at any given moment. This function should be defined as receiving the data model references through an argument, as they will be passed by the api automatically. When this function is used internally, there is a wrapper that gets the values from the vector being used in the optimization and updates the values in the data models, processing the defined necessary convertions.

5 Definition of the residuals

The residuals must be defined and set for the optimizer. In this context, a residual is a list with two elements, the first element is a string representing the residual, and the second element is a list of the parameters that influence the residual. The residual may then be added to an ordered dictionary contained in the optimizer (optimizer.*pushResidual*). This will allow for the printing and visualisation of the errors, as well as the creation of the sparse matrix. The definition of the residuals is closely related to the way the objective function is implemented and has to do with what error is being measured.

6 Definition of the sparse matrix

This matrix is also known as a Jacobian sparsity structure. It is useful in order to avoid the explicit computation of a Jacobian matrix (first-order partial derivatives) and instead use the finite difference approximation. The sparse matrix makes this process time feasible by marking elements which are known to be non-zero. The matrix contains information about the relationship between each of the parameters and the residuals. It is a matrix of '1's and '0's, where a '1' means the parameter of that column has an influence on the residual of that row and a '0' means it does not.

This is powerful in terms of computation complexity because it guides the optimizer in which parameters will influence each residual. It can be somewhat cumbersome to generate because of all the existing combinations of parameters and the need for the definition of their relationship with the residuals. However, because of the way the residuals are defined in `Step 5`, a function was implemented to derive the sparse matrix automatically. After the definition of the parameters and the residuals, the function optimizer.*computeSparseMatrix* may be called to extrapolate the sparse matrix.

7 Definition of the visualisation function

A custom visualisation function, fit for the problem, should be defined and set for the optimizer (optimizer.*setVisualizationFunction*). It is not required, but it is very useful for debugging, and in general, to try to understand if the optimization is working as intended. It serves as a display of the internal state of the optimization, along with a default graph implemented for OptimizationUtils, that shows error associated with each residual at the current and initial moments.

8 Starting the optimization

The optimization can be started by calling optimizer.*startOptimization*. Optimization options may be passed as arguments, such as tolerance of termination parameters and relative step size. During the optimization, along with the defined visualisation function, some OptimizationUtils functions are called that print out relevant information about the optimization and allow for the visualisation of the evolution of the error.

9 Outputting the final state

After the optimization is finished, the final state of the parameters may be saved to disk. What is to be saved and how it is done depends on the nature of the optimization and the context in which its results will be put to use.

Additional functionality

Some functionality of OptimizationUtils not mentioned includes: obtaining a list of all the available parameters and filtering it, particularly useful when defining the residuals; adding noise to the vector of parameters; and many different functions for printing out to the terminal, this includes information regarding the vector, models, residuals, etc. These last ones are instrumental for debugging purposes.

4.4 OPTIMIZATION CASE STUDY: OPTIMIZATION OF CAMERA POSES

The purpose of this case study was to apply the generic optimization approach outlined in section 4.3 to the problem of refining the poses of each camera with respect to a common reference frame.

A camera is defined by the sensor's internal properties and its positioning in space, relative to a reference frame. The former is represented by the intrinsic camera matrix, which was consistent since the same device was always used. It was retrieved using a camera calibration procedure and can be represented as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where c_x, c_y are the principal point coordinates, and f_x, f_y are the focal lengths expressed in pixel units. The latter is represented by an extrinsic camera matrix, such as:

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

which may be decomposed into two components, a translation and a rotation. The translation can be represented as the vector $T = (t_x, t_y, t_z)$, and the rotation can be represented as a 3×3 rotation matrix R.

The optimization is performed as a bundle adjustment, meaning the objective of the optimization is to refine the camera poses and the 3D point positions. As such, the set of parameters to be optimized $\mathbf{\Phi}$, is defined as:

$$\mathbf{\Phi} = \Big[\overbrace{x_{i=1}, y_{i=1}, z_{i=1}, r1_{i=1}, r2_{i=1}, r3_{i=1}, \ldots, x_{i=I}, y_{i=I}, z_{i=I}, r1_{i=I}, r2_{i=I}, r3_{i=I},}^{Camera\ poses} \overbrace{x_{j=1}, y_{j=1}, z_{j=1}, \ldots, x_{j=J}, y_{j=J}, z_{j=J}}^{Marker\ translations} \Big] \tag{4.3}$$

where i refers to the i-th camera, of the set of I cameras, and j refers to the j-th aruco marker, of the set of J aruco markers. Notice that, in this vector, the rotation for one camera (r_1, r_2, r_3) is represented through the axis/angle parameterization, as opposed to the 3×3 rotation matrix format of the camera's extrinsic matrix. Because a rotation matrix has $3 \times 3 = 9$ elements, but only 3 degrees of freedom, a different parameterization is necessary in order to intrinsically incorporate constraints on the rotations during the optimization.

Popular parameterization for rotations are Euler angles, quaternions, and axis/angle representation. However, not all representations are suitable for an optimization. Parameterization should not introduce more numerical sensitivity than the one inherent to the problem itself, as this decreases the chances of convergence in optimizations. When the parameterization

formats follow this rule, they can be referred to as fair parameterization [35]. For example, Euler angles, which are probably the most used angle parameterization, are not suitable for optimizations [36], because they do not yield smooth movements, each rotation is non-unique and, most notably, there are singularities, known as Gimbal lock, where one degree of freedom is lost [36]. Because quaternions have 4 components which are norm-1 constrained, and this introduces some complexity in the algorithms, they are usually not used for optimizations [36], even though they are a fair parameterization. The axis/angle parameterization is the most widely used to represent a rotation in an optimization. It is a fair parameterization and has only three components, two values to define an axis and one value to define an angle. In this way, any rotation can be represented as a rotation around this axis, by an angle θ.

To convert between the axis/angle format and the 3×3 rotation matrix format used in the data model to ease implementation, the Rodrigues' rotation formula is used. It provides an efficient method for computing the rotation matrix $R \in SO(3)$ corresponding to a rotation by an angle θ about a fixed axis specified by the unit vector $\hat{\omega} = (\omega_x, \omega_y, \omega_z) \in R^3$:

$$R = I\frac{sin\theta}{\theta}\tilde{\omega} + \frac{1 - cos\theta}{\theta^2}\tilde{\omega} \tag{4.4}$$

where I is the 3×3 identity matrix, and $\tilde{\omega}$ is defined as:

$$\tilde{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{4.5}$$

The starting poses of the cameras are given by the Open Constructor dataset. We also need a first guess for the position of the 3D points, which correspond to the aruco marker centers. Their poses must be determined in the world reference frame. Because the camera poses found in the Open Constructor dataset provide us with the transformation from each camera to the world, we only need a transformation from the aruco marker reference frame to one camera, which can be retrieved using the utilities of OpenCV's ArUco Library. Thus, the initial positions for the 3D points are obtained by, when going over the cameras to detect the aruco markers, whenever a marker is detected for the first time, calculating the aggregate transformation from the aruco marker to the world reference frame as:

$${}^{A_j}\mathbf{T}_W \;=\; {}^{C_i}\mathbf{T}_W \;\cdot\; {}^{A_j}\mathbf{T}_{C_i} \tag{4.6}$$

where A_j refers to the j-th aruco marker detected, C_i refers to the i-th camera (the first in which the j-th aruco marker can be detected), and W refers to the world reference frame. Then applying that transformation to the point corresponding to the center of the aruco marker in the marker's frame of reference. If the geometry of the scene were perfect, doing this for any camera that detected the aruco marker would result in the calculation of the same position.

The cost function is based on the reprojection error, i.e. the geometric error corresponding to the image distance between a projected point and a measured one, see section 2.4. The

relationship between a 3D point in the world and the pixels that correspond to its projection on an image plane can be expressed as:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}^{IntrinsicMatrix} \overbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}^{ExtrinsicMatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.7}$$

where X, Y, Z are the coordinates of a 3D point in the world; u, v are the coordinates of the projection point in pixels; c_x, c_y is the principal point; and f_x, f_y are the focal lengths expressed in pixel units. This explains how the changes to extrinsic parameters of a camera, which correspond to its pose, will have an influence on the error measured.

The optimization is performed using a nonlinear least-squares regression, as indicated for Bundle Adjustment problems. The function *least squares* from the SciPy library is used. It only requires the cost function, the vector of parameters, the bounds (infinite by default), and the sparse matrix (none used by default) to carry out the optimization. All of these things are set in place by the OptimizationUtils methods, simply by following the outline described in section 4.3. At the end of the optimization, the function returns a vector with the optimized parameters for the pose of the cameras and the position of the 3D points, which minimises the reprojection errors. Because of the wrapper implemented, these values are automatically converted and copied from this vector to their place in the data models.

The following is the direct concretization of the abstraction outlined in section 4.3 for camera pose estimation optimization:

1 The OCDatasetLoader, see section 4.1, was instantiated and used to load all the information from the dataset to memory, namely the images and transformations for each capture. The OCArucoDetector, see section 4.2, was used to detect and estimate the poses of the aruco markers, loading all related information to memory.

2 The optimizer was instantiated. The cameras' dataset and the aruco markers' dataset were each added to the optimizer as a data models (using the *addModelData* function).

3 A getter and a setter for the camera's translation matrix and a getter and a setter for the camera rotation matrix were defined. A getter and a setter for the marker's translation matrix were defined.

4 The error was computed as the Euclidean distance between the projected coordinates of the aruco marker's centers and the coordinates given by the detection of the aruco markers in each image. These distances correspond to the reprojection errors. As such, the cost function

f was defined as:

$$f = \sum_{i=1}^{N_d} \sqrt{\left(x^i{}_{\text{reproj}} - x^i{}_{\text{det}}\right)^2 + \left(y^i{}_{\text{reproj}} - y^i{}_{\text{det}}\right)^2} \tag{4.8}$$

where N_d is number of detections of camera-marker pairs, x_{reproj}, y_{reproj} are the coordinates of the reprojection of the 3D points, and x_{det}, y_{det} are the coordinates of the points detected in the image.

5 The residuals were identified as all detected camera-marker pairs. A string representing the residual ("C{cameraNumber}A{arucoID}") was obtained by, for each camera, going over the list of detected markers. Afterwards, the list of parameters that influence each of these pairs was defined, with the help of optimizer.*getParamsContainingPattern*. This function takes advantage of naming conventions to extrapolate, from the complete list of parameters (translation and rotation of each camera and translation of each marker), only the ones related to a specific residual (camera-marker pair). Finally, a list containing the two elements (residual representation and list of parameters that influence it) is added to the optimizer using optimizer.*pushResidual*.

6 The function optimizer.*computeSparseMatrix* is called to calculate the sparse matrix. Figure 4.2 shows a fragment of the sparse matrix that would be generated for a couple of cameras, with a few detections each. The missing columns should have the rest of the parameters for camera 0 (C000r2 and C000r3), all the parameters for camera 1, and all the parameters for the rest of the aruco markers detected in the scene by any camera. The naming convention used for the parameters were "C{cameraNumber}{parameter}" for the camera parameters and "A{arucoID}{parameter}" for the aruco marker parameters.

	$C000_{tx}$	$C000_{ty}$	$C000_{tz}$	$C000_{r1}$	...	$A504_{tx}$	$A504_{ty}$	$A504_{tz}$
$C000A529$	1	1	1	1	...	0	0	0
$C000A527$	1	1	1	1	...	0	0	0
$C000A555$	1	1	1	1	...	0	0	0
$C000A412$	1	1	1	1	...	0	0	0
$C000A536$	1	1	1	1	...	0	0	0
$C000A504$	1	1	1	1	...	1	1	1
$C001A529$	0	0	0	0	...	0	0	0
$C001A527$	0	0	0	0	...	0	0	0
$C001A412$	0	0	0	0	...	0	0	0
$C001A536$	0	0	0	0	...	0	0	0
$C001A504$	0	0	0	0	...	1	1	1
$C001A554$	0	0	0	0	...	0	0	0

Figure 4.2: Fragment of a sparse matrix.

7 The custom visualisation function was defined and set for the optimizer (optimizer.*setVisualizationFunction*). It consists of showing all the images, each with the aruco markers centers detected, the initial projections connected to the target by a blue line, and the projections at the current step of the optimization, as seen in Figure 4.4. This is complemented

by a 3D representation of the pose of the cameras and position of the 3D points in the scene, showcased in Figure 4.5. The default visualization function is shown in Figure 4.3.

8 The optimization is started (optimizer.*startOptimization*).

9 After the optimization is finished, the final state of the models is saved to disk. This is described in detail in subsection 4.4.1.

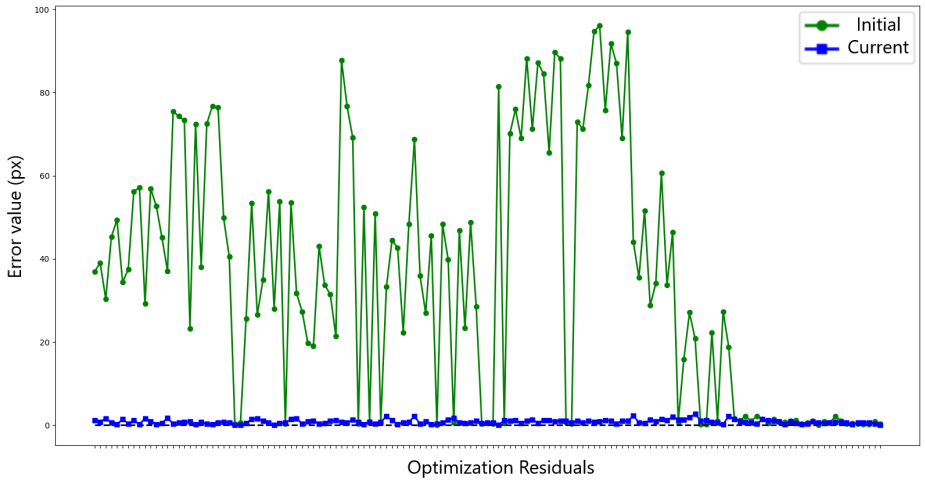

Figure 4.3: Default visualisation function of OptimizationUtils: Graph of the value of error associated with each residual. Initial reprojection error in green, current reprojection error in blue.

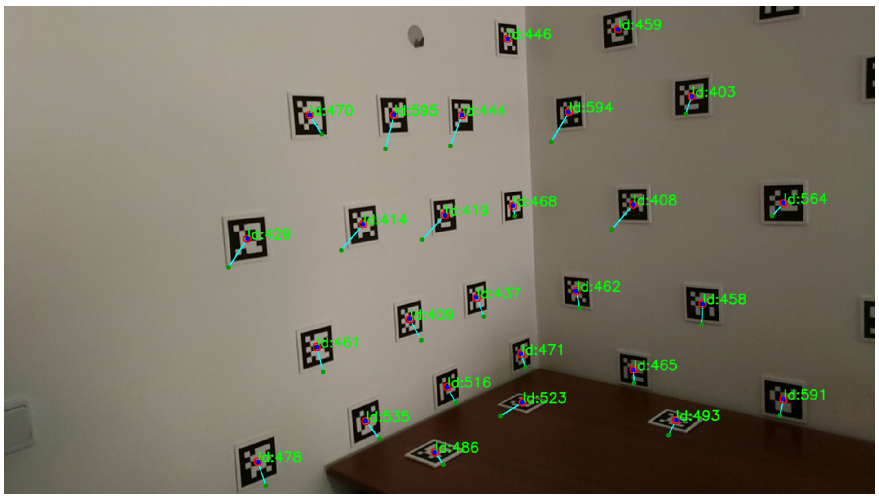

Figure 4.4: Example of reprojection visualisation for one of the images of the dataset. Aruco marker detection in red (target of optimization), initial projection in green, current projection in dark blue.

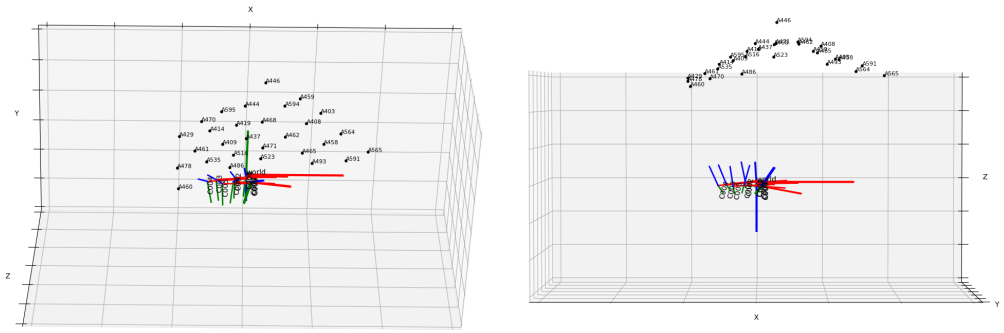

Figure 4.5: 3D representation of the position of the cameras and aruco markers in the scene, produced by the visualisation function.

4.4.1 TO OUTPUT A DATASET

In this particular problem, it was considered that the best way to output the result of the optimization would be to create a new, optimized dataset, in Open Constructor fashion. In order to accomplish this, the folder must be created programmatically and all the images are copied from the original dataset. The .txt files for each camera must be written to the new folder, these contain transformations from the world to the camera, from the world to the depth sensor, and from the world to the device. Next, the point clouds must be written to .ply files in the same folder. Each point cloud is read from its original .ply file and the transformation from the old world coordinates to new world coordinates is applied ("old" meaning before optimization and "new" meaning after optimization). Effectively, this is a transformation from the world to the camera frame of reference, saved from before optimization, followed by a transformation from the camera to the world, obtained after the optimization:

$$^{\text{oldW}_i}\mathbf{T}_{\text{newW}} = {}^{\text{C}_i}\mathbf{T}_{\text{newW}} \cdot {}^{\text{oldW}}\mathbf{T}_{\text{C}_i} \tag{4.9}$$

where W refers to World and C_i refers to the i-th camera. At this point, the transformation calculated previously is applied to each of the point clouds and they are saved to separate .ply files, along with a colour selected from a colour map. The function of this colour is to help differentiate the point clouds when they are analysed together. Since we want to clearly distinguish the colours of point clouds next to each other, a qualitative colour map was chosen. The colour map tab10 was imported from Matplotlib, see Figure 4.6. Finally, one point cloud composed of all the original point clouds merged, and one composed of all the optimized point clouds merged, are created. This is to facilitate the handling of the clouds for result analysis.

Figure 4.6: Matplotlib's qualitative colormap tab10.

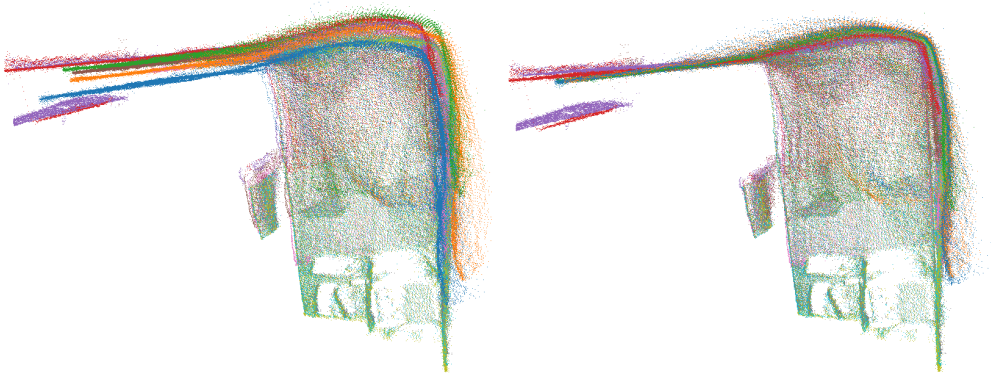

Figure 4.7: Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 1). Before optimization on the left, after optimization on the right.

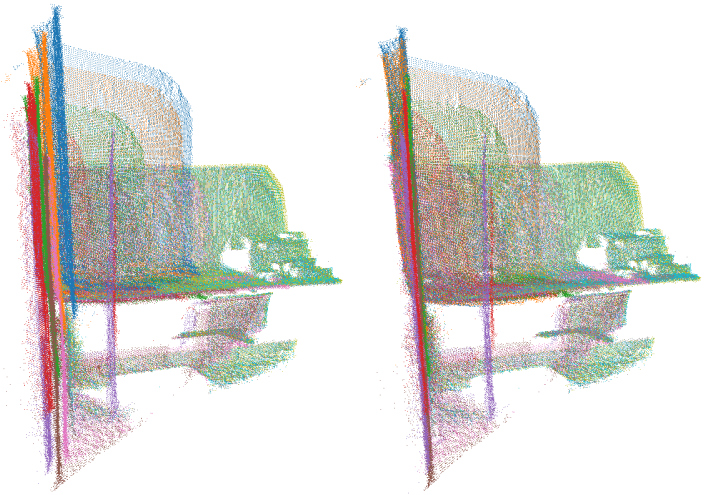

Figure 4.8: Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 2). Before optimization on the left, after optimization on the right.

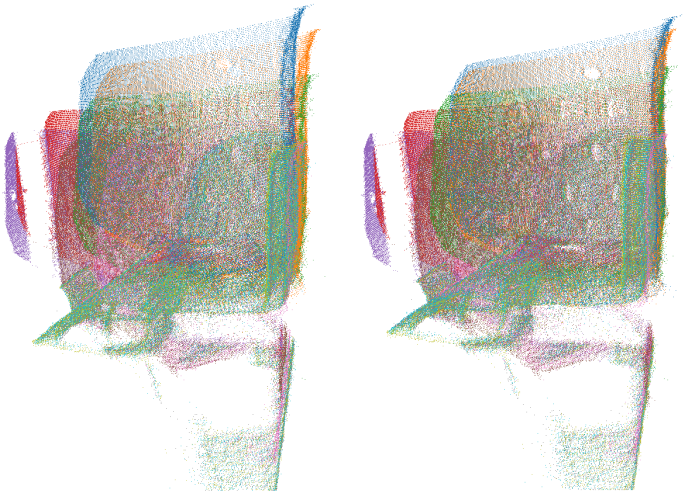

Figure 4.9: Optimization of OfficeDemo result comparison using Matplotlib's tab10 colormap (view 3). Before optimization on the left, after optimization on the right.

CHAPTER 5

Methodology for Fiducial Marker Removal

This chapter focuses on a module developed to automatically remove fiducial markers from the texture of a scene. In section 4.4, a method to enhance the registration of a 3D reconstruction using fiducial markers was presented. The module showcased in the present chapter was created as an attempt to avoid the pollution of the acquired scene's texture with these markers. There are other scenarios, besides 3D reconstruction, where having fiducial markers in the environment is advantageous, such as Augmented Reality (AR) applications, and being able to remove them may improve the final product significantly.

This module relies on the OCDatasetLoader to provide access to the data, namely transformations, point clouds and corresponding images. This means that the code is agnostic to the way the dataset is stored and loaded to memory. By using a different version of a dataset loader, one may utilise this inpainting tool in a different context. The tool was programmed to detect and remove aruco markers, since this was the kind of fiducial marker used in section 4.4. However, it could be used for different markers by adapting the detection and pose estimation functions.

ArucoInpaintingTool contains the implementation to automatically process RGB-D data, removing aruco markers from the RGB images and projecting the information to point clouds. Works with OCDatasetLoader, see section 4.1, and OCArucoDetector, see section 4.2, to remove aruco markers from the texture of a 3D model saved in OpenConstructor dataset format.

The starting point for the removal of the markers from the texture of the scene was the decision to use an inpainting algorithm, more specifically the Navier-Stokes-based Inpainting from OpenCV's *inpaint*. Besides a few parameters, this function requires a mask for the areas to be inpainted. Initially, the masks used were the ones obtained by detecting the marker corners in the image.

All the masks showcased in this chapter were blended with images to better showcase where they fit and what regions are the target.

5.1 CREATION OF THE MASKS

In order to use the OpenCV *inpaint* function, which is intended to restore a selected region in an image using the region neighbourhood, one must create a mask in order to define this region. The plan was to remove the fiducial markers from the texture of the scene, and as such, their very detection was used to create these masks. This can be observed in Figure 5.1.

The problem with this is that the are detected only includes symbol itself, not accounting for the margin around it. This creates a mask that would not allow for the inpainting of the whole item, leaving blank cards scattered through the scene. The first attempt to solve this problem was to dilate the masks, using OpenCV's *dilate* function, as shown in Figure 5.2. Being a 2D operation, this didn't work very well. For instance, markers an angle and closer to the camera need a bigger mask dilation, possibly in different directions.

The solution found for this was to create a 3D object with the same measurements as the markers, including thickness, so the masking would work even when the angle towards the camera is so large that it is possible to see the sides of the marker. This object is made up of only the vertices. It must be transformed from each detected marker's reference frame to the camera's reference frame and then projected to a 2D image. This is accomplished by using the OpenCV function *projectPoints* with the translation and rotation vectors associated with each marker (and the intrinsic parameters and distortion coefficients associated with the camera). Then, taking advantage of the created function *drawMask*, it is possible to draw the vertices and fill in the figure. These masks are showcased in Figure 5.3. A representation of the difference observed is shown in Figure 5.4.

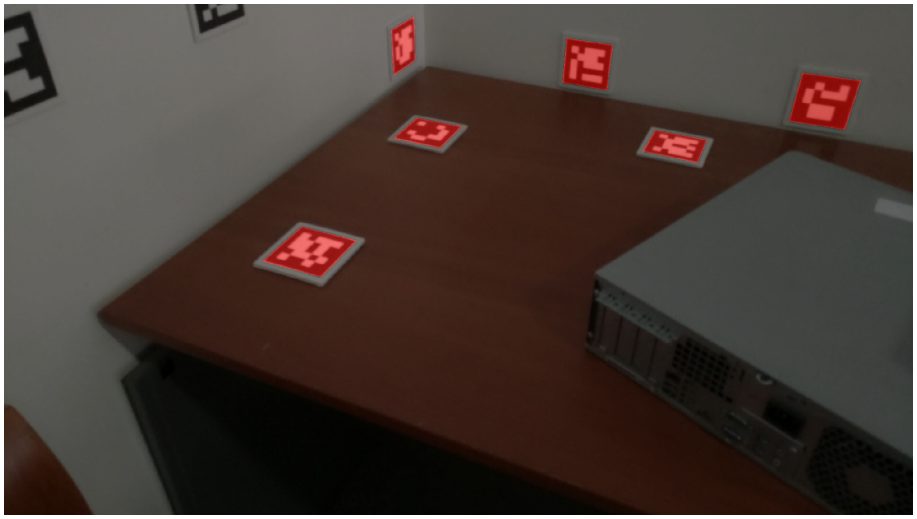

Figure 5.1: Simple mask created from fiducial marker detection.

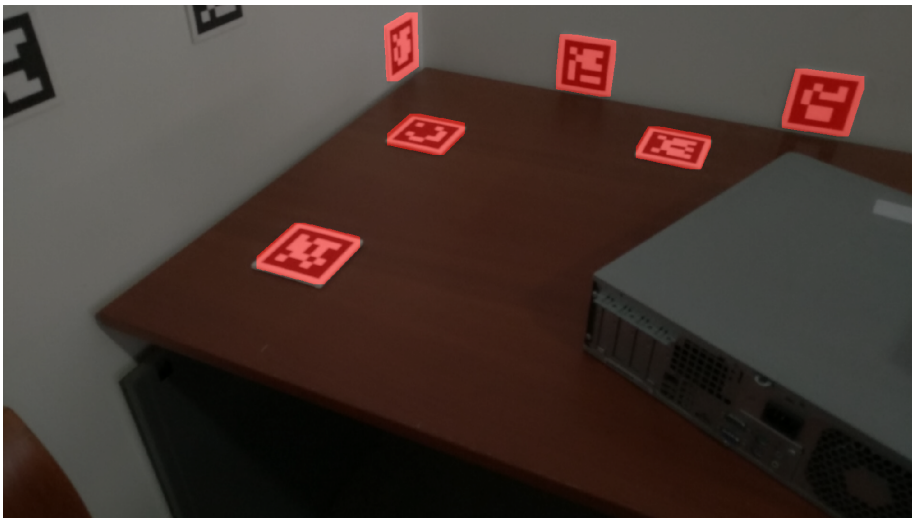

Figure 5.2: Mask after dilation was applied.

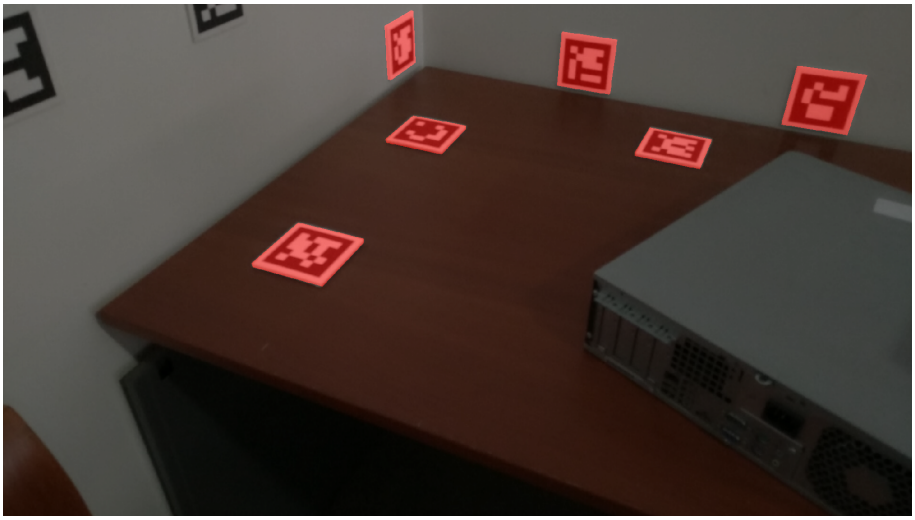

Figure 5.3: Mask created from the projection of the 3D object.

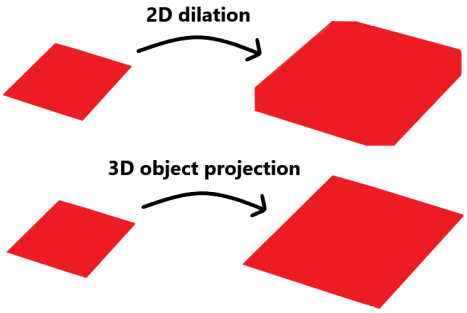

Figure 5.4: Results for 2D dilation and 3D projection methods for mask creation.

5.2 NAVIER-STOKES-BASED INPAINTING

The first step was to create an accurate representation of the region covered by the marker, see Figure 5.5. Afterwards, the challenge was to define the colours which will be used to restore that region. The results of the Navier-Stokes-based Inpainting from OpenCV's *inpaint* were underwhelming, as shown in Figure 5.6. The examples found online for the testing of this function are usually about removing lines from the images or restoring damage from folding photographs. Inpainting in this manner is usually done in small areas, while the size of a fiducial marker in the scene is substantial.

Figure 5.5: Mask used for marker removal.

Figure 5.6: Navier-Stokes-based inpainting from `OpenCV` *inpaint*.

5.3 IMPROVING THE INPAINTING

5.3.1 FIRST STAGE

In order to improve the result of the inpainting, blurring the area where the inpainting took place seemed like a good idea. To accomplish this, the image where the inpainting was applied was copied and blurred using the OpenCV function *medianBlur* with a high aperture linear size, as seen in Figure 5.8. Then, the mask used to perform the inpainting was used to replace the pixels in those areas of the original image, with the pixels of the blurred image, see Figure 5.7. The result is showcased in Figure 5.9.

Figure 5.7: Mask for the pixels which will be replaced by the ones in Figure 5.8.

Figure 5.8: Auxiliar image 1. Median blur of Figure 5.6.

Figure 5.9: First stage result.

5.3.2 SECOND STAGE

The overall texture obtained in subsection 5.3.1 looks better than the original inpainting restoration. However, the limits of where the texture taken from the blurred image was applied are sharp and visible. To reduce this effect, a new set of masks was created. This takes advantage of the method described in section 5.1, transforming and projecting a 3D object with the shape of the marker, yet making the area of the square somewhat larger, see Figure 5.10. The image obtained in the first stage is copied and blurred using a median blur again, however this time with a smaller aperture linear size, to avoid mixing colour of the surrounding objects, resulting in Figure 5.11. This new mask is, much like before, used to copy the corresponding regions from the blurred image to the original first stage result, creating Figure 5.12.

Figure 5.10: Mask for the pixels which will be replaced by the ones in Figure 5.11.

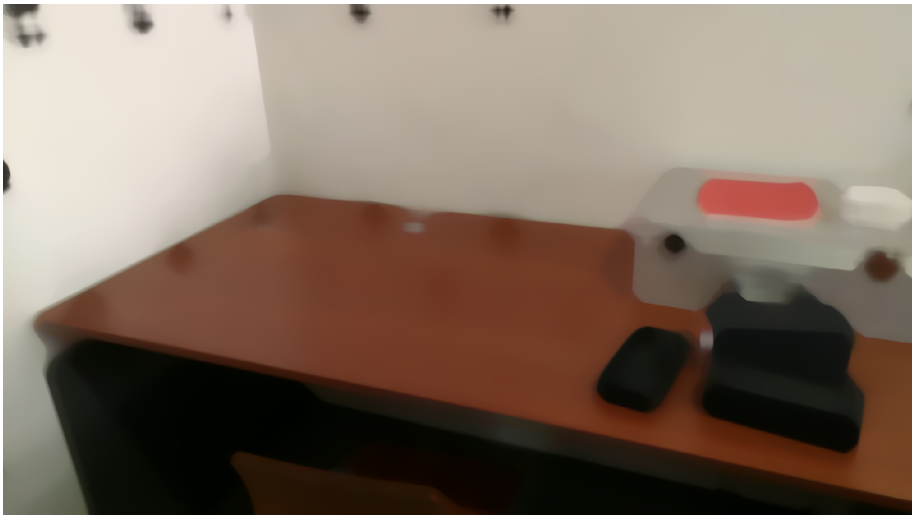

Figure 5.11: Auxiliar image 2. Median blur of the first stage result.

Figure 5.12: Second stage result.

5.3.3 FINAL RESULTS

The process is finished by applying a Bilateral Filtering function (OpenCV's *bilateralFilter*). This allows for the preservation of the edges in the image and avoids the mixing of colours, while smoothing over the surfaces, making the grainy pattern of the walls a little less noticeable for example, which makes the inpainted areas blend in a little better. The final result of the whole inpainting process is showcased in Figure 5.14, which may be compared to the original image in Figure 5.13.

Figure 5.13: Original image from OfficeDemo dataset.

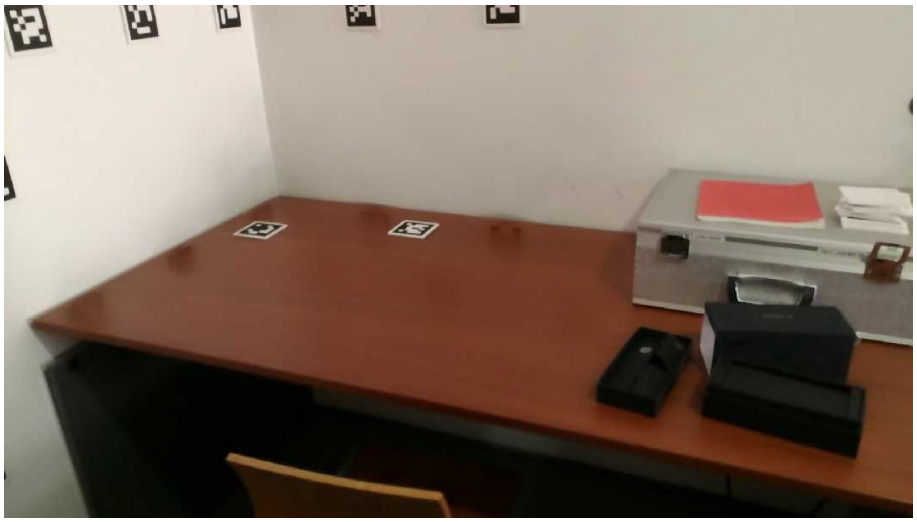

Figure 5.14: Restored image from OfficeDemo dataset.

5.4 INPAINTING NON-DETECTED MARKERS: CROSS-INPAINTING

It is a frequent occurrence that some fiducial markers in the pictures of a dataset are not detected. This may be because the marker is not fully in the field of vision of the camera, because the camera was in motion and the picture taken was blurred, or simply because the marker is at a very awkward angle. Since the transformations from the aruco markers to cameras and from the cameras to the world are known, see Figure 5.15, it was theorised that it should be possible to know if a marker can be found within an image, even if it wasn't detected in it, for the sake of removing it from the texture.

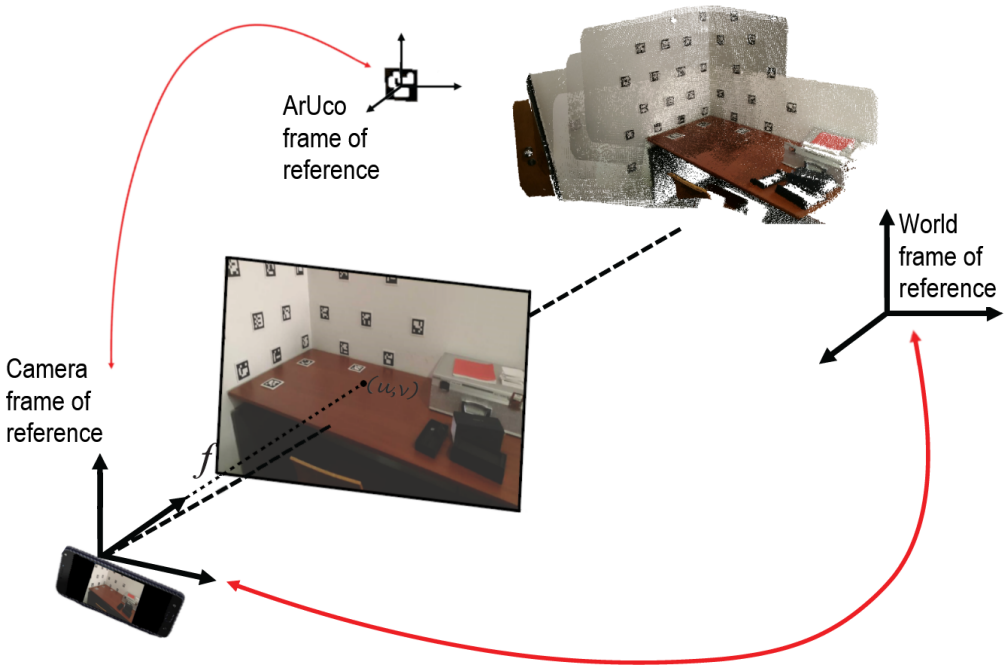

Figure 5.15: Illustration of conversion between different reference frames.

The solution implemented, named "Cross-Inpainting", takes advantage of transformations from the aruco markers to the world. Every time a new marker is detected in one of the images of the dataset, the transformation of the marker to the world is calculated and saved. This transformation is calculated using the transformation of the marker to the camera where it was detected, the i-th camera, and the transformation from that camera to the world:

$$ {}^{\mathrm{A}_j}\mathbf{T}_{\mathrm{W}} = {}^{\mathrm{C}_i}\mathbf{T}_{\mathrm{W}} \cdot {}^{\mathrm{A}_j}\mathbf{T}_{\mathrm{C}_i} \tag{5.1} $$

where A_j refers to the j-th aruco marker detected, C_i refers to the i-th camera, and W refers to the world reference frame. Given an image captured by a camera k, when this image is being inpainted, it is possible to access the information for all markers. It is checked if there are any markers that can be projected within the image, but that have not been detected in it. This is done by applying the transformation from the aruco marker reference frame to the

world, as in Equation 5.1, followed by the transformation from the world to camera k:

$$ {}^{\mathrm{A}_j}\mathbf{T}_{\mathrm{C}_k} = {}^{\mathrm{W}}\mathbf{T}_{\mathrm{C}_k} \cdot {}^{\mathrm{A}_j}\mathbf{T}_{\mathrm{W}} \tag{5.2} $$

where A_j refers to the j-th aruco marker, C_k refers to the k-th camera, and W refers to the world reference frame. The 3D marker object is transformed using Equation 5.2 and then projected to the image captured by camera k. An example of what is obtained from these projections can be seen in Figure 5.16. The algorithm appears to work, but the poses of the cameras and the positions of the markers in relation to the world do not match, and as such, the regions covered by the fiducial markers are not matched by the masks, see Figure 5.18. However, after an optimization is performed, the placement of the masks is significantly enhanced, as seen in Figure 5.17, and the results improve accordingly, see Figure 5.19.

Figure 5.16: Final mask before optimization. Masks obtained from aruco marker detection in red and masks obtained from projections from other cameras (Cross-Inpainting) in blue.

Figure 5.17: Final mask after optimization. Masks obtained from aruco marker detection in red and masks obtained from projections from other cameras (Cross-Inpainting) in blue.

Figure 5.18: Restoration result before optimization.

Figure 5.19: Restoration result after optimization.

5.5 3D COLOUR VISUALISATION

In order to better visualise the point clouds and make them closer to a final product it would be interesting to have them coloured, particularly with images now free of markers. To achieve this, one has to read the information from the .ply extension files found in the dataset, see section 3.2, project the 3D points to the images, and extract the RGB value of the corresponding pixels. The coordinates of the 3D points are found in the world reference frame, however it is stored in an OpenGL coordinate system. Thus, the points must be read from the file, converted from OpenGL to the OpenCV coordinate system, transformed from the world to the camera's reference frame and projected to the image. The RGB value can then be extracted. Finally, a new .ply extension file is created and the information of colour is stored along with the original vertices.

When all of this is done, and the `.PLY` file is saved correctly, the content of this file can

be visualised using Meshlab or some `CloudViewer` class, the `PCL` library for instance or a simple viewer provided by the `PPTK` library, and the output should be something similar to Figure 5.20, where the restoration of the texture can be observed before and after the optimization.

Figure 5.20: Point cloud coloured with texture obtained from restored images, using cross-inpainting. Before optimization on the left and after optimization on the right (meshlab).

For comparison purposes, in Figure 5.21 a merged point cloud, coloured with images inpainted using only the detected markers can be observed. At first glance, the texture applied before optimization seems better, this effect is caused by the fact that the points closest to the viewer happen to correspond to images where more markers were detected. After the optimization, the merged point clouds are better aligned and, as such, the markers that were occluded before show through.

Figure 5.21: Point cloud coloured with texture obtained from restored images, no cross-inpainting. Before optimization on the left and after optimization on the right (meshlab).

In Figure 5.22 we can see a merged point cloud coloured with the original image. The enhancement of the registration caused by the optimization improves the texture significantly,

as showcased by Figure 5.23.

Figure 5.22: Merged point cloud coloured with texture obtained from images. Before optimization on the left and after optimization on the right (meshlab).

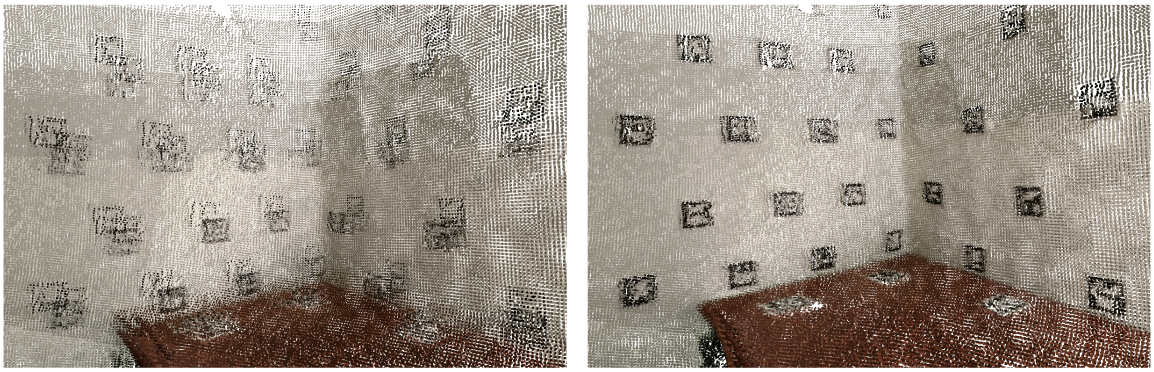

Figure 5.23: Merged point cloud coloured with texture obtained from images (detail). Before optimization on the left and after optimization on the right (meshlab).

CHAPTER 6

Results

This chapter contains the evaluation methodology and the description of its process, going over the collection of datasets and showcasing the results obtained in various ways.

6.1 Evaluation Methodology

Visual comparison of the point clouds before and after optimization may sometimes prove to be somewhat subjective and inconclusive. Particularly in datasets with a very large number of point clouds. The reporting of results in a document also means that the changes must be observed in a side by side manner, which makes them harder to perceive than in Meshlab for instance, see section 3.2, where it is possible to instantly switch between point clouds placed in the same exact place. A solution for this would be to have a ground truth. We use a laser scan of a meeting room in Departamento de Engenharia Mecânica (DEM) at University of Aveiro. This room, being a lot bigger than the one used for the datasets created previously, was also an opportunity to create a more difficult challenge for the optimizer. Unfortunately, the equipment needed for the laser scan was not available at the same time as the Zenphone AR, and as such, the datasets and the ground truth could not be captured in the same day, resulting in some physical differences in the scene.

6.2 Collecting Datasets

The datasets were collected using the ZenFone AR, described in chapter 3. The OfficeDemo dataset was collected in office 22.2.18 of Departamento de Engenharia Mecânica, Universidade de Aveiro. It is a relatively small dataset with a large density of fiducial markers. One larger, more complex dataset was captured within the meeting room mentioned in section 6.1, and one more was taken out in the lobby, just outside the meeting room.

As far as the meeting room dataset collection goes, and its comparison to the FARO laser scan, as mentioned before, they could not be collected in the same day, and as such, a number of items had been moved within the room, such as couches and chairs. Thus, a decision was made to capture the area that had been changed the less, focusing on one corner where the furniture hadn't been moved. Regardless, there were still some noticeable differences.

6.3 FARO DATASET

The laser scan of the meeting room was acquired with a FARO Focus Laser Scanner and the point clouds for what would become the ground truth had to be exported using the FARO Scene software.

A single point cloud was obtained from the merging of several scans in Cloud Compare, see section 3.2. However, it was very dense and, as such, computationally hard to work with. Because it was unnecessarily dense for its purpose, it was downsampled and clipped, also using Cloud Compare. The final result can be observed in Figure 6.2.

Figure 6.1: FARO generated point cloud after downsampling.

Figure 6.2: FARO generated point cloud after downsampling and clipping.

6.4 EVALUATION PROCEDURE

The key metric used for the evaluation procedure was the Hausdorff distance. It measures how far two subsets of a metric space are from each other, and is often applied to the measurement of distance between point clouds. It is defined as the greatest of all the distances from a point in one set to the closest point in the other set. The version implemented in Meshlab, see section 3.2, allows us to know, not only an absolute distance, but the distance at each point of the cloud visually through the application of a colormap.

ICP was used in Cloud Compare to transform point clouds to the same reference frame as the ground truth point cloud. Afterwards, Meshlab's Haudorff distance was calculated between the original point clouds and the ground truth point cloud, and the per vertex quality function is used to apply a colormap, making it visually understandable. The same process is then carried out between the optimized point clouds and the ground truth point cloud. It is a red-yellow-green-blue colormap, where red codes the smallest distance and blue the greatest.

This way, visually it is possible to understand if parts of the scene may be impacted negatively by the improvement of other areas. It is also useful to observe if the differences between the Tango generated datasets and the FARO dataset are because of error in reconstruction or just because the scene had physically changed between the capturing of the ground truth and our datasets.

6.5 MEETINGROOM DATASET

In this section, the results obtained using the MeetingRoom dataset are showcased. This dataset takes advantage of the fact that there is the possibility of analysing it in comparison with the FARO generated dataset section 6.3. The results are first compared in a before-and-after manner, using the texture of the original images in Figure 6.3, Figure 6.5 and Figure 6.4, then the Hausdorff distance based comparison is discussed, and finally, the fiducial marker removal results are showcased in Figure 6.8.

6.5.1 OPTIMIZATION OF CAMERA POSES RESULTS

This subsection contains the results for the optimization of camera poses in the MeetingRoom dataset.

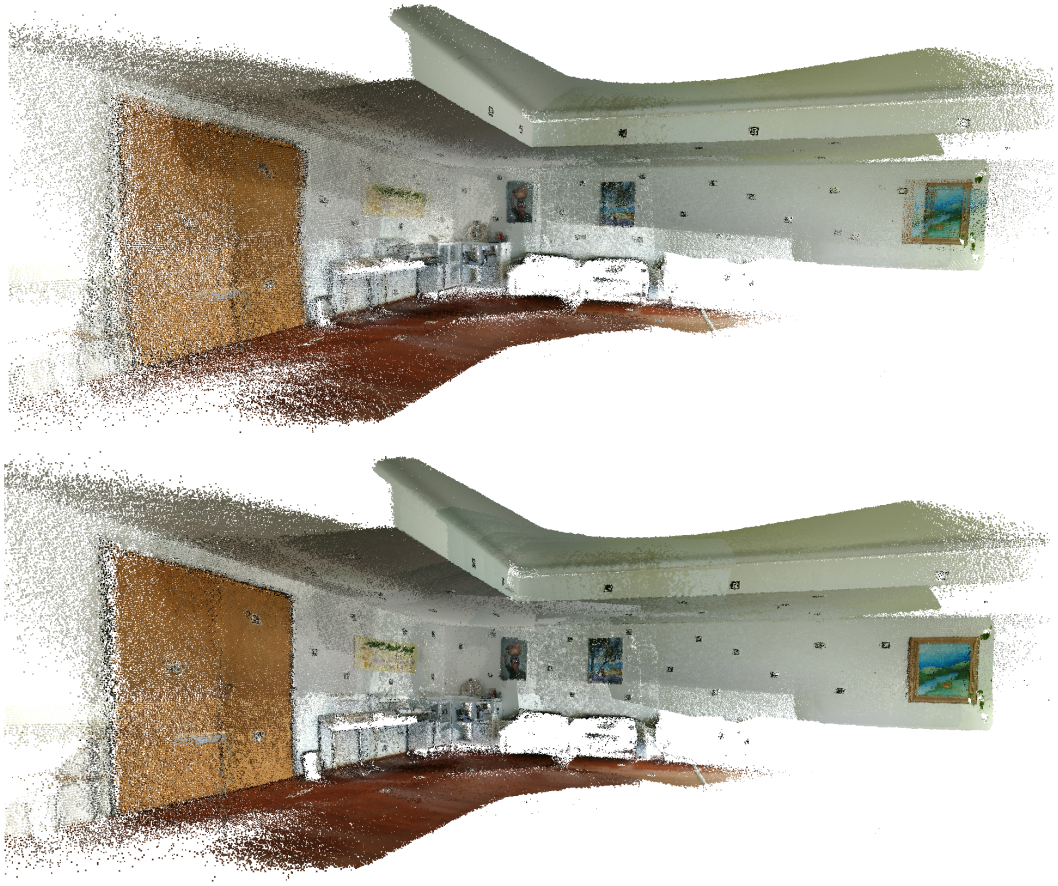

Figure 6.3: MeetingRoom dataset coloured with texture obtained from original images. Before optimization on the top and after optimization on the bottom.

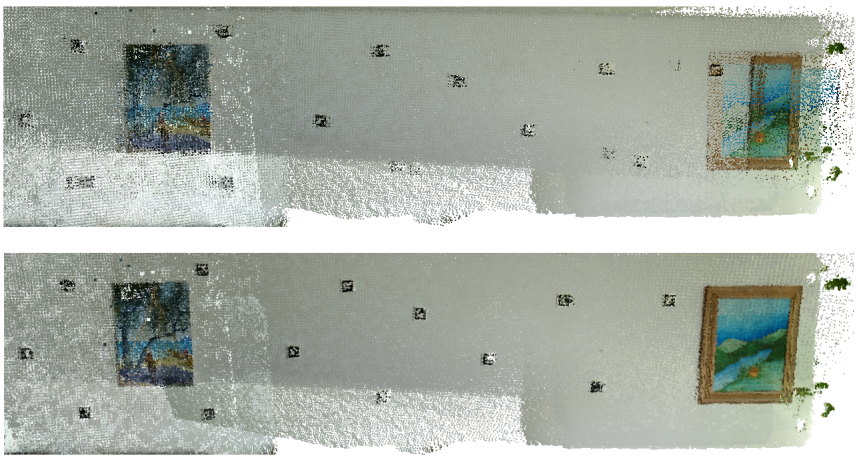

Figure 6.4: MeetingRoom dataset coloured with texture obtained from original images (detail 1). Before optimization on the top and after optimization on the bottom.

Figure 6.5: MeetingRoom dataset coloured with texture obtained from original images (detail 2). Before optimization on the top and after optimization on the bottom.

The Hausdorff distance was calculated from the MeetingRoom dataset to the FARO dataset, before and after optimization. The local values were used to colormap the point clouds, see Figure 6.6 and Figure 6.7. The mean and Root Mean Square (RMS) values, see Table 6.1, indicate an improvement of roughly 0.009 m and 0.012 m respectively, which corresponds to approximately 27%. The maximum distance considered for the measurement was 150cm, determined as a good limit by trial and error method. It is hard to place these numbers into context, because the merged point clouds contain a very large number of points, many of which were not moved significantly during the optimization. These points correspond to areas already close to the ground truth before the optimization. As a result, areas that were improved may not have a very high impact on the value of the mean and Root Mean Square (RMS).

	mean (m)	**RMS (m)**	**min (m)**	**max (m)**
Before Optimization	0.0337	0.0442	0.0001	0.1500
After Optimization	0.0247	0.0324	0.0001	0.1500

Table 6.1: Hausdorff distance from the MeetingRoom dataset to the FARO dataset.

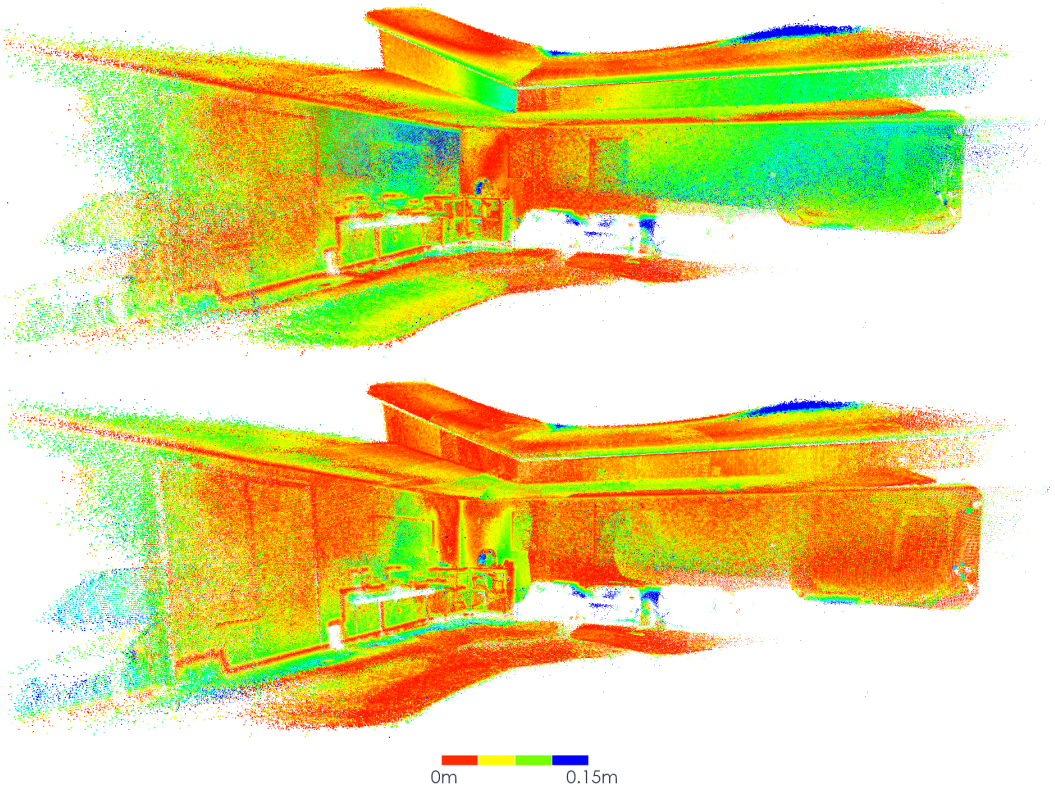

Figure 6.6: MeetingRoom dataset colormapped with Hausdorff distance to FARO dataset. Before optimization on the top and after optimization on the bottom.

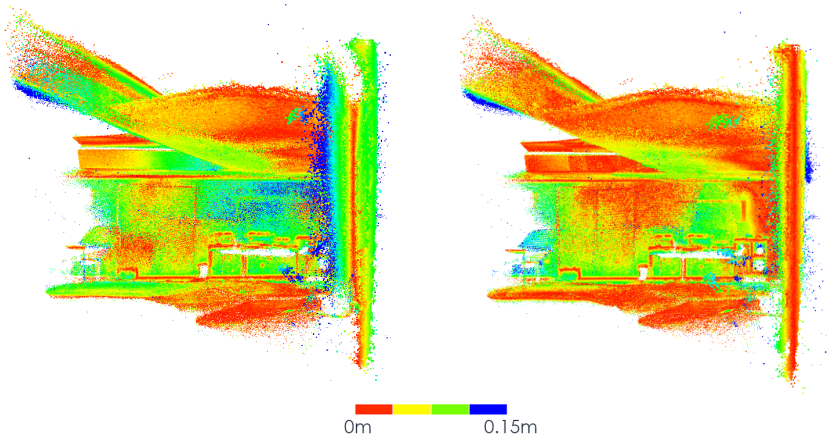

Figure 6.7: MeetingRoom dataset colormapped with Hausdorff distance to FARO dataset (detail). Before optimization on the left and after optimization on the right.

6.5.2 FIDUCIAL MARKER REMOVAL RESULTS

This subsection contains the results for texture application and fiducial marker removal in the MeetingRoom dataset.

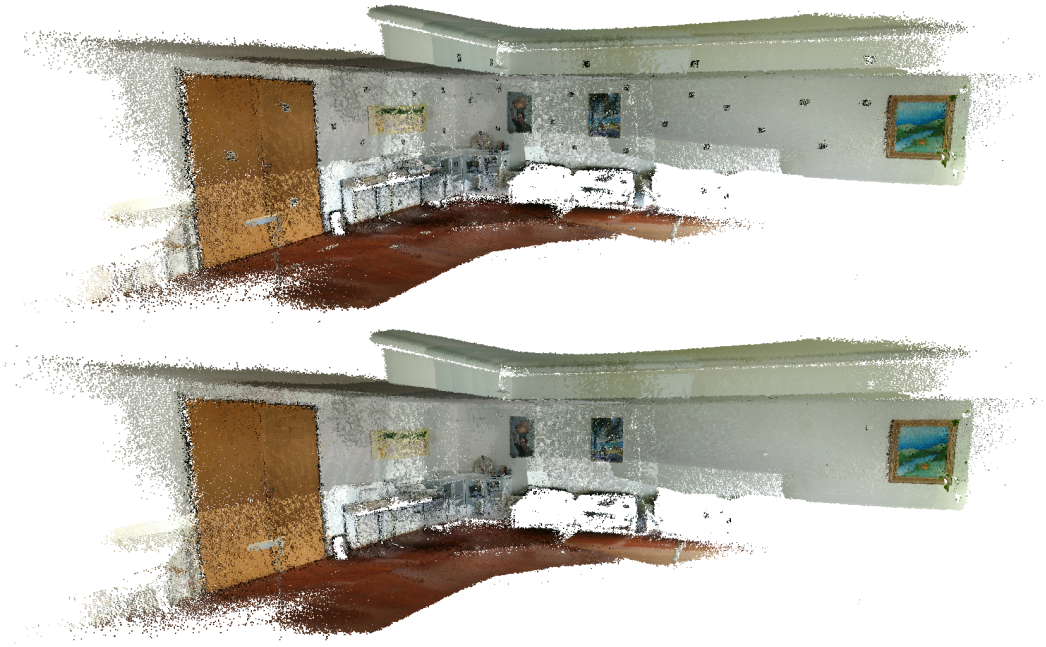

Figure 6.8: MeetingRoom dataset coloured with texture obtained from original images on the top and texture obtained from restored images on the bottom.

6.6 LOBBY DATASET

In this section, the results obtained using the Lobby dataset are showcased. This dataset is analysed through visual before-and-after comparison, starting with the texture of the original images in Figure 6.9 and Figure 6.10, then showing some detail using Matplotlib's tab10 colormap, see Figure 6.11, and finally showcasing the fiducial marker removal results in Figure 6.12.

6.6.1 OPTIMIZATION OF CAMERA POSES RESULTS

This subsection contains the results for the optimization of camera poses in the Lobby dataset.

Figure 6.9: Lobby dataset coloured with texture obtained from original images. Before optimization on the top and after optimization on the bottom.

Figure 6.10: Lobby dataset coloured with texture obtained from original images (detail). Before optimization on the left and after optimization on the right.

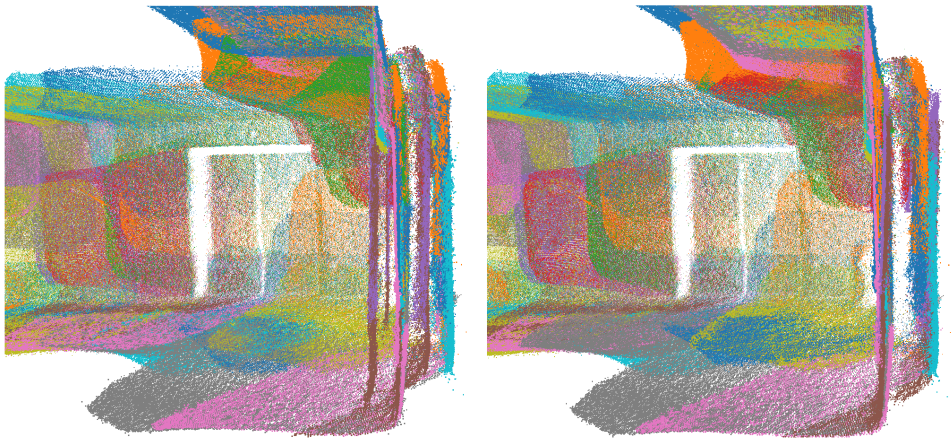

Figure 6.11: Lobby dataset detail coloured with Matplotlib's tab10 colormap. Before optimization on the left and after optimization on the right.

6.6.2 FIDUCIAL MARKER REMOVAL RESULTS

This subsection contains the results for texture application and fiducial marker removal in the Lobby dataset.

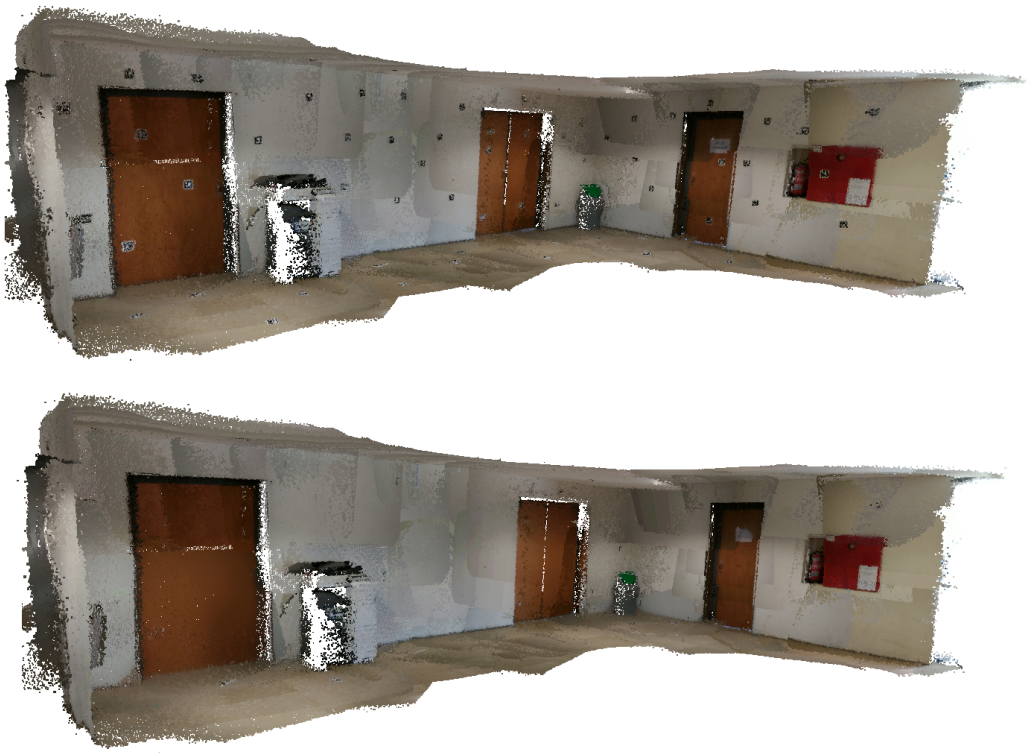

Figure 6.12: Lobby dataset coloured with texture obtained from original images on the top, and texture obtained from restored images on the bottom.

CHAPTER 7

Conclusions

It is possible to observe an improvement in the geometry of the 3D reconstructions that stems from an enhancement in the alignment of the point clouds. Therefore, it can be concluded that the optimizer is working as intended. Nevertheless, there are some limitations, derived from the fact that this optimization only targets the registration problem, while having no influence on the geometry of each point cloud. In other words, the alignment of the point clouds may be improved, but if the point clouds are themselves distorted, which may occur with RGB-D data, the geometry of the scene may not improve significantly from this optimization.

The optimization implemented attempts to improve the results obtained from a 3D reconstruction using RGB-D cameras. If the dataset obtained from the reconstruction has poor alignment of the point clouds, even if only on some areas, then we should expect the optimization to improve the scene's geometry significantly. However, if a particular dataset happens to have very good alignment, the improvement will probably prove to be minor.

The optimization was implemented making use of the tools developed and the created code structure, described in section 4.3. This base api provides useful abstractions and an environment for future implementation of optimizations. It facilitates building upon and future work, allowing for the writing of more intuitive code. It also provides structure for a systematic approach to this kind of problem.

The removal of the fiducial markers from the texture of the scene was achieved. The technique developed to complement the OpenCV inpainting function was a success. Although, it is important to note that it was created for restoring areas without a recognisable or sharp pattern. Employing the arucoInpaintingTool, one can produce point clouds with a texture that is free of markers, by utilising RGB-D information. The cross-inpainting technique was successful at removing non-detected aruco markers from the images, as long as there is good registration of the point clouds. For this reason, the technique also serves as a way to corroborate the observation that the optimization is working as expected, since its results, which depend on the good alignment of the point clouds, improve after the optimization is performed, as observed in section 5.5.

Point clouds were considered more interesting than meshes as the product of the 3D reconstructions for this work. This is because the creation of meshes would cause some information to be lost. Information that could prove important for the analysis of the results of the optimization. However, for future work, it would be interesting to generate meshes from the final point clouds, coloured with the texture free of fiducial markers. These meshes may be created through the implementation of ROS-based CHISEL, for instance. The work on the development of this tool was started, however it wasn't given priority, and was put on hold indefinitely.

There are some doubts about the ideal density of fiducial markers in the scene, i.e. how many markers should should be detected, on average, in each image, to obtain the best results from the optimization. In the future, this could be analysed, producing some guidelines to make the placement of fiducial markers more efficient.

Regarding the removal of fiducial markers, it would be interesting to research techniques that would allow the restoration of texture containing distinctive patterns, to enable its use in more different types of environments.

References

[1] H. Fan, W. Yao, and Q. Fu, «Segmentation of Sloped Roofs from Airborne LiDAR Point Clouds Using Ridge-Based Hierarchical Decomposition», *Remote Sensing*, vol. 6, pp. 3284–3301, Apr. 2014. DOI: 10.3390/rs6043284.

[2] A. Henn, G. Gröger, V. Stroh, and L. Plümer, «Model driven reconstruction of roofs from sparse LIDAR point clouds», *International Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 17–29, Feb. 2013. DOI: 10.1016/j.isprsjprs.2012.11.004.

[3] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, «KinectFusion: Real-time dense surface mapping and tracking», in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Oct. 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[4] J. Han, L. Shao, D. Xu, and J. Shotton, «Enhanced Computer Vision With Microsoft Kinect Sensor: A Review», *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, Oct. 2013, ISSN: 2168-2267. DOI: 10.1109/TCYB.2013.2265378.

[5] I. Afanasyev, A. Sagitov, and E. Magid, «ROS-Based SLAM for a Gazebo-Simulated Mobile Robot in Image-Based 3D Model of Indoor Environment», in *Advanced Concepts for Intelligent Vision Systems*, S. Battiato, J. Blanc-Talon, G. Gallo, W. Philips, D. Popescu, and P. Scheunders, Eds., Cham: Springer International Publishing, 2015, pp. 273–283, ISBN: 978-3-319-25903-1.

[6] M. Westoby, J. Brasington, N. Glasser, M. Hambrey, and J. Reynolds, «'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications», *Geomorphology*, vol. 179, pp. 300–314, Dec. 2012. DOI: 10.1016/j.geomorph.2012.08.021.

[7] A. Diakité and S. Zlatanova, «FIRST EXPERIMENTS WITH THE TANGO TABLET FOR INDOOR SCANNING», *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-4, pp. 67–72, Jun. 2016. DOI: 10.5194/isprsannals-III-4-67-2016.

[8] L. Li, F. Su, F. Yang, H. Zhu, D. Li, Z. Xinkai, F. Li, Y. Liu, and S. Ying, «Reconstruction of Three-Dimensional (3D) Indoor Interiors with Multiple Stories via Comprehensive Segmentation», *Remote Sensing*, Aug. 2018. DOI: 10.3390/rs10081281.

[9] Y. Zhou, X. Zheng, R. Chen, X. Hanjiang, and S. Guo, «Image-Based Localization Aided Indoor Pedestrian Trajectory Estimation Using Smartphones», *Sensors*, vol. 18, p. 258, Jan. 2018. DOI: 10.3390/s18010258.

[10] A. Hermans, G. Floros, and B. Leibe, «Dense 3D semantic mapping of indoor scenes from RGB-D images», May 2014, pp. 2631–2638. DOI: 10.1109/ICRA.2014.6907236.

[11] A. Jamali, P. Boguslawski, and A. Abdul Rahman, «A Hybrid 3D Indoor Space Model», *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W1, pp. 75–80, Oct. 2016. DOI: 10.5194/isprs-archives-XLII-2-W1-75-2016.

[12] M. Zollhöfer, A. G. Patrick Stotko, C. Theobalt, M. Nießner, R. Klein, and A. Kolb, «State of the Art on 3D Reconstruction with RGB-D Cameras», 2018. [Online]. Available: https://web.stanford.edu/~zollhoef/papers/EG18_RecoSTAR/paper.pdf.

[13] S. Gokturk, H. Yalcin, and C. Bamji, «A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions», Jul. 2004, pp. 35–35. DOI: 10.1109/CVPR.2004.17.

[14] M. Minou, T. Kanade, and T. Sakai, «METHOD OF TIME-CODED PARALLEL PLANES OF LIGHT FOR DEPTH MEASUREMENT.», pp. 521–528, Aug. 1981.

[15] P. Will and K. Pennington, «Grid coding: A preprocessing technique for robot and machine vision», *Artificial Intelligence*, vol. 2, no. 3, pp. 319–329, 1971, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(71)90015-4. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370271900154.

[16] B. Curless and M. Levoy, «A Volumetric Method for Building Complex Models from Range Images», in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96, New York, NY, USA: ACM, 1996, pp. 303–312, ISBN: 0-89791-746-4. DOI: 10.1145/237170.237269. [Online]. Available: http://doi.acm.org/10.1145/237170.237269.

[17] S. Rusinkiewicz, O. Hall-holt, and M. Levoy, «Real-Time 3D Model Acquisition», *ACM Transactions on Graphics*, vol. 21, May 2002. DOI: 10.1145/566570.566600.

[18] J. Minguez, L. Montesano, and F. Lamiraux, «Metric-based iterative closest point scan matching for sensor displacement estimation», *Robotics, IEEE Transactions on*, vol. 22, pp. 1047–1054, Nov. 2006. DOI: 10.1109/TRO.2006.878961.

[19] B. Manjunath, C. Shekhar, and R. Chellappa, «A New Approach to Image Feature Detection With Applications», *Pattern Recognition*, vol. 29, pp. 627–640, Apr. 1996. DOI: 10.1016/0031-3203(95)00115-8.

[20] D. Lowe, «Distinctive Image Features from Scale-Invariant Keypoints», *International Journal of Computer Vision*, vol. 60, pp. 91–, Nov. 2004. DOI: 10.1023/B:VISI.0000029664.99615.94.

[21] H. Bay, T. Tuytelaars, and L. Van Gool, «SURF: Speeded Up Robust Features.», vol. 110, Jan. 2006, pp. 404–417.

[22] M. J. A. Patwary, S. Parvin, and S. Akter, «Significant HOG-Histogram of Oriented Gradient Feature Selection for Human Detection», *International Journal of Computer Applications*, vol. 132, pp. 20–24, Dec. 2015. DOI: 10.5120/ijca2015907704.

[23] A. Babaryka, «Recognition from collections of local features», *Master of Science Thesis Stockholm*, 2012.

[24] D. Mount, N. Netanyahu, and J. Le Moigne, «Efficient Algorithms for Robust Feature Matching», *Pattern Recognition*, vol. 32, Apr. 2003. DOI: 10.1016/S0031-3203(98)00086-7.

[25] C.-S. Chen, Y.-P. Hung, and J.-B. Cheng, «RANSAC-Based DARCES: A new approach to fast automatic registration of partially overlapping range images», *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, pp. 1229–1234, Dec. 1999. DOI: 10.1109/34.809117.

[26] J. Martínez, J. González-Jiménez, J. Morales, A. Mandow, and A. Garcia, «Mobile robot motion estimation by 2D scan matching with genetic and iterative closest point algorithms», *Journal of Field Robotics*, vol. 23, pp. 21–34, Jan. 2006. DOI: 10.1002/rob.20104.

[27] S. Thrun and J. J. Leonard, «Simultaneous Localization and Mapping», *Springer Handbook of Robotics*, pp. 871–889, 2008. DOI: 10.1007/978-3-540-30301-5_38.

[28] S. J. W. Jorge Nocedal, «Numerical optimization», *Springer*, 2000.

[29] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, «Bundle Adjustment in the Large», Nov. 2010, pp. 29–42. DOI: 10.1007/978-3-642-15552-9_3.

[30] A. Harltey and A. Zisserman, *Multiple view geometry in computer vision (2. ed.)*. Jan. 2006, ISBN: 978-0-521-54051-3.

[31] F. Romero Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, «Speeded Up Detection of Squared Fiducial Markers», *Image and Vision Computing*, vol. 76, Jun. 2018. DOI: 10.1016/j.imavis.2018.05.004.

[32] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, «Generation of fiducial marker dictionaries using Mixed Integer Linear Programming», *Pattern Recognition*, vol. 51, Oct. 2015. DOI: 10.1016/j.patcog.2015.09.023.

[33] S. Weiss, M. Achtelik, S. Lynen, M. Chli, and R. Siegwart, «Real-time Onboard Visual-Inertial State Estimation and Self-Calibration of MAVs in Unknown Environments», May 2012. DOI: 10.1109/ICRA.2012.6225147.

[34] «Inertial measurement unit», 4 711 125, Dec. 1987.

[35] J. Hornegger and C. Tomasi, «Representation issues in the ML estimation of camera motion», vol. 1, Feb. 1999, 640–647 vol.1, ISBN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.791285.

[36] J. Schmidt and H. Niemann, «Using Quaternions for Parametrizing 3-D Rotations in Unconstrained Nonlinear Optimization», Jan. 2001.