FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Web Service for Automatic Generation of 3D Models through Video

Pedro Costa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Teixeira

July 20, 2015

Web Service for Automatic Generation of 3D Models through Video

Pedro Costa

Mestrado Integrado em Engenharia Informática e Computação

July 20, 2015

Abstract

This thesis addresses the problem of obtaining the structure of a scene and the motion of a camera, also known as Structure from Motion, taking a video as input. Structure from Motion has a wide range of applications and, although it is a well studied area, there is still not a lot of commercial Business to Consumer applications. This work tries to shift the focus towards increasing the dissemination and usability of this type of applications, while taking into account the time and cost constraints.

To tackle the existing problems, the video is acquired by a monocular acquisition system. This allows for any regular video camera, such as the camera of a *smartphone*, to be used to perform the reconstruction. Nonetheless, the greatest contribution of this work, is the creation of a scalable distributed system that performs the reconstruction and can be exposed as a *web service*. This service allows other developers to create their own applications, even for lesser powerful devices, provided that they have an Internet connection.

This project had to solve two separate problems: a Computer Vision problem, where an existing Structure from Motion method had to be improved in order to be parallelized, and a Distributed Systems problem, where a scalable system had to be implemented tackling the problem of sharing large amounts of data between nodes.

The Computer Vision problem was solved by dividing the input video, into several smaller videos, and performing the Structure from Motion individually. Then, all the reconstructions are aligned and merged. Some properties of the reconstruction are used in order to reduce the effects of noise on the video. The method is evaluated using a synthetic dataset and videos acquired from a *smartphone*. The method was proven to be robust to noise over long video sequences.

The Distributed Systems problem was solved by using the BitTorrent protocol to transfer frames between nodes and a message broker to communicate between them. Medium sized files, such as matrices, are simply transferred over TCP.

Resumo

Esta dissertação aborda o problema de obter a estrutura da cena e movimento da câmara, também conhecido como *Structure from Motion*, tendo um vídeo como entrada. A área de *Structure from Motion* tem um grande número de aplicações e, apesar de ser uma área bastante estudada, ainda não existem muitas aplicações comerciais *Business to Consumer*. Este trabalho tenta mudar o foco para aumentar a disseminação e usabilidade deste tipo de aplicações, ao mesmo tempo que tem em consideração restrições de tempo e custo.

Para resolver os problemas existentes, o vídeo é obtido por um sistema de aquisição monocular. Desta forma, qualquer câmara de filmar, como por exemplo a câmara de um *smartphone*, pode ser usada para efetuar a reconstrução. No entanto, a maior contribuição deste trabalho é a criação de um sistema distribuído escalável que efetua a reconstrução e pode ser exposto como um serviço *web*. Este serviço permite que outros programadores criem as suas próprias aplicações, mesmo para dispositivos menos poderosos, desde que tenham ligação à Internet.

Este projeto necessitou de resolver dois problemas distintos: um problema de Visão por Computador, onde um método existente de *Structure from Motion* teve que ser melhorado de forma a ser paralelizado, e um problema de Sistemas Distribuídos, onde um sistema escalável teve que ser implementado resolvendo o problema de partilhar grandes quantidades de dados entre nós.

O problema de Visão por Computador foi resolvido dividindo o vídeo de entrada em vídeos mais pequenos, e aplicando o método de *Structure from Motion* individualmente. Depois, todas as reconstruções são alinhadas e fundidas. Algumas propriedades da reconstrução são usadas de forma a reduzir os efeitos do ruído no vídeo. O método foi avaliado usando um *dataset* sintético e vídeos obtidos por um *smartphone*. Provou-se que o método é robusto ao ruído em vídeos longos.

O problema de Sistemas Distribuídos foi resolvido utilizando o protocolo do BitTorrent para transferir *frames* entre nós e um *message broker* para comunicar entre eles. Ficheiros de tamanho intermédio, tais como matrizes, são transferidos utilizando TCP.

Acknowledgements

First of all, I want to thank my supervisor, professor Luís Teixeira, for his readiness to answer all my questions and for his advices. Without his valuable input this thesis would not be possible.

I wish to express my sincere thanks to Sebastian Bukowiec, from CERN, for the important suggestions regarding the distributed system.

I take this opportunity to express gratitude for professor Rui Rodrigues for providing me with access to the Computer Graphics laboratory and to the 3D Scanner. I also want to thank Hugo Machado and Francisco Martins for helping me handling the 3D Scanner.

I am grateful to professor Rui Nóbrega for his encouragement and suggestions.

Finally, I thank my parents for supporting me and for giving me the opportunity of enrolling on a Master of Science degree. I am also grateful to my girlfriend for all the love and patience along this venture.

Pedro Costa

"Only the wisest and stupidest
of men never change."

Confucius

Contents

List of Figures

List of Tables

Abbreviations

ADCC	Algorithm to Detect Critical Configurations
API	Application Programming Interface
ATE	Absolute Translation Error
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
CAD	Computer Aided Design
CPU	Central Processing Unit
CENSURE	Center Surround Extremas
DoF	Degrees of Freedom
DoG	Difference of Gaussian
DoH	Determinant of Hessian
FAST	Features from Accelerated Segment Test
FIFO	First In First Out
FIRST	Fast Invariant to Rotation and Scale Transform
GPCA	Generalized Principal Component Analysis
GPS	Global Positioning System
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
ISVD	Iterative Singular Value Decomposition
IVT	Incremental Visual Tracker
KLT	KanadeLucasTomasi
LIDAR	Light Detection And Ranging
NRSfM	Non-rigid Structure from Motion
ORB	Oriented BRIEF
PCA	Principal Component Analysis
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
RMSE	Root Mean Squared Error
RPE	Relative Pose Error
RPFM	Robust Parallel Factorization Method
SfM	Structure from Motion
SIFT	Scale Invariant Feature Detector
SLAM	Simultaneous Localization and Mapping
SLAT	Simultaneous Localization and Tracking
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator

Chapter 1

Introduction

The generation of a 3D model of a scene through 2D images is a fundamental task in Computer Vision. This type of software has applications in several areas, such as: image-based 3D modelling, hand-eye calibration, augmented reality, autonomous navigation, motion capture, image and video processing, remote sensing, photo organization and browsing, segmentation and recognition and military applications [WKYW13].

Image-based 3D modelling is of particular interest for researchers due to its wide applicability and low cost. While creating simple 3D models using CAD tools is relatively easy, obtaining an accurate model of a complex real scene or object remains difficult. By using videos or images to generate 3D models of real scenes or objects, the time and costs of manual modelling are reduced.

Structure from Motion (SfM) is the most known technique for performing image-based 3D modelling. While this has been an active area of research for decades, it remains active due to a few breakthroughs and more practical applications. New algorithms and applications of SfM continue being developed every year [Sze10].

1.1 Motivation

The main reasons behind the increase in interest for image-based techniques are both economical and technical. For instance, Google's autonomous vehicles have about $150,000 in equipment that includes a $70,000 LIDAR (laser radar) system[1]. Such high-priced equipments would increase the final price of the car, making this technology available only for the most luxurious cars.

On the other hand, costs of cameras have steadily decreased over the past years, even for the industrial cameras that produce high quality, high frame rate and are robust to extreme conditions. Cameras are ideal for SfM tasks, not only due to their reduced price, but also because they can capture textures, that enrich the final model and can be used to improve the reconstruction.

[1] http://tinyurl.com/n9rwuqb

In fact, most people living in developed countries have easy access to a monocular camera. According to a study from eMarketer[2], by 2016 there will be more than 2 billion *smartphone* users. This year (2015), and for the first time, one quarter of the global population will have a *smartphone*.

Most SfM applications are used for urban reconstruction but can also be used to create models for video-games, movies, and even 3D printers. These applications typically use additional sensors or stereoscopic cameras however, algorithms directed to monocular cameras already achieve good results.

Commercial applications that resort to 3D reconstruction began being developed in the past years, for example Microsoft's Photosynth[3] that reconstructs a scene using a set of photos given by the user, or Autodesk's 123D Catch[4] that uses a set of photos to create a 3D model of an object, that can be 3D printed. Other applications use a video captured from a stereoscopic or structured light sensor in order to perform the reconstruction of an object or scene. An example of these applications is the ReconstructMe[5] that uses, among other sensors, Microsoft's *Kinect* to perform the reconstruction in real time.

Despite the great effort that has been done in making these applications more usable, there is still room for improvements. ReconstructMe uses a sensor that is wired to the computer along the whole reconstruction process and that require previous calibration. Also, most users do not possess such a sensor and are required to buy one. As the reconstruction is performed on the user's computer, it might not comply with the hardware requirements.

The other applications require a set of photographs of the object to perform the reconstruction. These approaches struggle with textureless objects and may require a considerable amount of photographs to perform satisfactorily.

This thesis ultimately aims to turn SfM applications more usable to the end user. The user is allowed to film the object with a monocular camera, such as the camera of a *smartphone*, instead of taking photos. This approach is more usable, since it is faster and more convenient for the user. By taking a video, instead of photographs, there is also more information to use.

Similarly to the 123D Catch, a *web service* is created, making the reconstruction available from every device, provided that they have an internet connection. As the *web service* can be exposed, other developers can use the service to build new applications, abstracting all the process of 3D reconstruction.

Such applications would allow the creation of avatars by simply using the web-cam. It would also allow the user to customize its games or augmented reality applications with real world objects. Also, with the rise of home 3D printers, a user would be able to replicate one of its objects without knowing how to use CAD tools.

One of the challenges of the monocular systems is that it is not possible to extract depth from a single frame, unlike stereoscopic systems. Nonetheless, stereoscopic systems need previous

[2] http://tinyurl.com/kkpxevo
[3] https://photosynth.net/
[4] http://www.123dapp.com/catch
[5] http://reconstructme.net/

calibration, while it is possible to perform auto-calibration on the monocular ones [PSC+04]. This is a desirable characteristic as it makes the system simpler and more flexible.

1.2 Scope

This work has two problems to solve:

1. **Computer Vision**: The creation of a parallel method to perform Structure from Motion using video.
2. **Distributed System**: The creation of a system that distributes the aforementioned method.

The Computer Vision problem was solved first, followed by the Distributed System problem as it was dependent of the former. More specifically, as Computer Vision is a broad area, this thesis is inserted in the rigid-body Structure from Motion area.

1.3 Contributions

Most Computer Vision algorithms are not distributed. The biggest contribution of this thesis is the construction of a system that is inherently distributed. A parallelizable method for Structure from Motion that is robust to noise over long sequences was developed. Then, a scalable distributed system, that is able to scale with respect to the number of users and to the number of frames is presented.

During the thesis, a paper was written, titled "Robust Parallel Factorization Method for Structure and Motion". This work was submitted to the 26th British Machine Vision Conference (BMVC) and, to the date of the writing, the authors were still waiting for the decision.

1.4 Document Structure

The structure of this thesis is as follows:

- **Chapter 1. Introduction:** The scope of the work, motivations as well as the proposed solution and main contributions.
- **Chapter 2. Literature Review:** A review of the work developed in the areas of Structure from Motion, Feature Matching, Point Cloud Reconstruction and Distributed Algorithms.
- **Chapter 3. Robust Parallel Factorization:** Review of the Factorization method and presentation of the developed method. An evaluation of the method was also performed in this chapter.
- **Chapter 4. Distributed Architecture:** The architecture of the distributed system that implements the Robust Parallel Factorization Method.

- **Chapter 5. Conclusions and Future Work:** Final conclusions and future directions.

Chapter 2

Literature Review

This thesis builds upon several other projects and research proposals that made this work possible and, therefore, some relevant areas are reviewed in this Chapter. Some of the reviewed methods are used, others are left out as future work and others are simply different approaches that are presented as comparison.

This Chapter is divided in 4 Sections, each one addressing a different area. Section 2.1 reviews the area of SfM, presents the work done with Factorization methods, how to refine the final reconstruction with Bundle Adjustment and the Scale Drift problem that affects monocular systems. Section 2.2 addresses the problem of matching points across different images. Section 2.3 presents the approaches that are used to create a 3D model from a point-cloud. Finally, Section 2.4 was dedicated to distributed algorithms, mainly in the context of computer vision.

Some sections of this chapter are based on some review works [COC+13, Rad10, THZ13, TMHF00, TM08, FS12] and Szeliski's book [Sze10].

2.1 Structure From Motion

Structure from Motion is the technique of simultaneously estimating the structure of the scene and pose of the camera. There are three main approaches to perform SfM for rigid objects: to measure 3D coordinates for surface points using equipments such as laser or structured light, stereo vision and factorization algorithms.

Wei et al. [WKYW13] states that the typical pipeline for SfM has several steps: detecting image features, matching features across images, estimating structure of the scene and the poses of the cameras and, finally, an optional bundle adjustment.

Pollefeys [PSC+04] described a complete system to perform SfM that was able to deal with uncalibrated image sequences with an hand-held camera. By performing a multi-view stereo matching, the author was able to obtain a dense estimation of the surface geometry. The author was able to introduce virtual objects into the video sequence after performing the reconstruction.

More recently, Agarwal et al. [AFS+09] were able to perform a reconstruction of city-scale image collections, with more than a thousand images, in less than a day. In order to be able to do that, the system uses distributed computer vision algorithms for image matching and 3D reconstruction, that maximize the parallelism at each stage of the pipeline, and scale with the number of images and with the amount of available computation.

One popular scenario in which SfM is commonly used is reconstruction in urban environments. Pylvänäinen et al. [PBK+12] used LIDAR scans, an IMU and a GPS, in conjunction with panorama images, to automatically create compact and accurate 3D city models for Augmented Reality applications. Sinha et al. [SSS+08] used a collection of unordered images to perform a realistic reconstruction of architectural structures and urban scenes. The system requires the user to outline the object to be reconstructed in order to obtain better performance.

These approaches use the stereo vision approach, some combined with additional equipments for better accuracy. The factorization approach is going to be reviewed more thoroughly in the next subsection.

2.1.1 Factorization

The factorization algorithm was first proposed by Tomasi and Kanade [TK92] assuming an orthographic camera model. The main idea of the method is to factorize the tracking matrix into motion and structure matrices using Singular Value Decomposition (SVD). This method is reviewed in more detail in section 3.1.

The limitation of assuming an orthographic camera model was later overcome by Poelman and Kanade [PK94] that developed an extension of the original method to assume a paraperspective model by imposing additional constraints to the system. Later, Triggs [Tri96] extended the method for the projective model, by recovering a set of projective depths using fundamental matrices and epipoles estimated from the measures. Han and Kanade [HK02] were able to perform Euclidean reconstruction, under the assumption of a perspective model, with uncalibrated cameras. A projective reconstruction is performed and, then, the reconstruction is converted to an Euclidean one by enforcing metric constraints.

One of the problems of the classical factorization method is that all points are required to be present on every frame. Tomasi and Kanade [TK92] overcame that problem by performing the reconstruction over a submatrix of the full measurement matrix and then, as they called it, *hallucinate* the position of the remaining points and pose of the camera on the remaining frames. Li et al. [LGD10] developed an algorithm to detect critical configurations (ADCC) in the video in order to optimize the division of a long sequence for the factorization method.

Hajder [Haj05] proposed an improvement to the original factorization method, using an iterative algorithm to refine the structure and motion data independently by minimizing an error function. Tests on synthetic data showed that the method performed significantly better for objects that consist of few points.

Other approaches use the Iterative Singular Value Decomposition (ISVD) [BN78] to perform online SfM [KBWT14], by updating the Structure and Motion matrices as soon as a frame arrives for processing. Masiero et al. [MGVP14] performed Euclidean reconstruction on low cost Android smartphones by combining ISVD with its sensors.

Buchanan and Fitzgibbon [BF05] developed a method that is able to factorize a matrix even in the presence of missing values. With such method there is no need for the *hallucination* step. More recently, Meng and De la Torre [MdlT13] were able to create a factorization method that makes no assumption on the model of the noise, by modeling it as a Mixture of Gaussians.

The factorization approach is the most common framework for Non-rigid Structure from Motion (NRSfM) [ZSXW15, DM15, XCK04, TYAB01]. The measurement matrix is assumed to have a greater rank and the object can be approximated using a linear combination of 3D basis shapes.

2.1.2 Bundle Adjustment

Bundle adjustment is the problem of refining a visual reconstruction in order to produce optimal estimates of the extrinsic and intrinsic parameters of the camera, as well as the 3D structure of the scene. In order to find the optimal parameters, a robust non linear cost function is minimized where the reprojection errors are estimated. The name refers to the "bundles" of light that come out of the 3D point and converge at the centre of each camera and, later, are "adjusted" in an optimal way with respect to its 3D position and position of the camera. This process is almost always used as the last step on all 3D reconstruction algorithms.

Triggs et al. [TMHF00] states that the bundle adjustment is formulated as a non linear least squares problem where the cost function is quadratic with the features' reprojection error, and the robustness is provided by an outlier detection. Pollefeys et al. [PSC+04] noted that, if the errors on the points can be modelled as a Gaussian distribution with zero mean, the bundle adjustment corresponds to a maximum likelihood estimator.

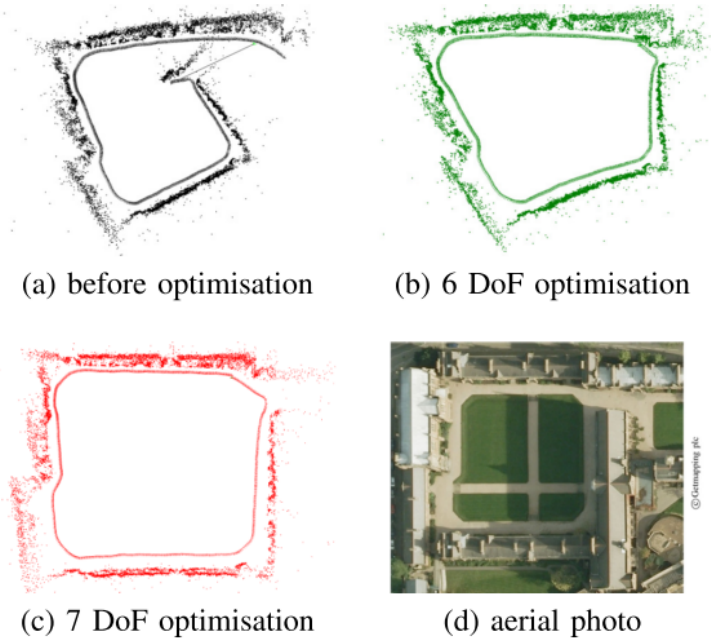
As a simplification, it is assumed that the scene is modelled by p 3D points X_p, seen in i images with camera pose and calibration P_i, and that features $\underline{x}_{ip}$ were obtained from each image i and point p. Beyond that, for each observation $\underline{x}_{ip}$ it is assumed that a predictive model $x_{ip} = x(P_i, X_p)$ is known. In the case of observing images or videos, this predictive model consists on reprojection, however, in other cases, measures obtained by sensors can be added. The cost function can be represented in the following manner:

$$\Delta x_{ip}(P_i, X_p) = \underline{x}_{ip} - x(P_i, X_p) \tag{2.1}$$

The problem of minimizing the function of Equation 2.1 is typically solved using the algorithm of *Levenberg Marquardt* [NW99].

2.1.3 Scale Drift Correction

Scale drift is a crucial challenge that prevents monocular systems from having the same performance as stereo systems.



(a) before optimisation (b) 6 DoF optimisation

(c) 7 DoF optimisation (d) aerial photo

Figure 2.1: Results before and after the *loop closure* [SMD10]. (a) The map calculated without *loop closure*. (b) An optimization of the graph using 6 DoF but without being able to solve the *drift* in the scale. (c) Results obtained by the optimization proposed by Strasdat et al. using 7 DoF. (d) An aerial image of the trajectory.

A possible approach to solve this problem is using *loop closure* [BDw06]. Events such as observing a point that was not visible for several frames or go back to an area previously mapped are called *loop closure*. These events can be detected by evaluating visual similarities between the current and past frames.

Angeli et al. [AFDM08] proposed a method based in visual words to detect *loop closure*. The authors showed that it is extremely fast to evaluate visual similarities by using an inverted index.

After the detection of the loop, it is possible to create a pose graph, where the nodes are the poses of the camera and the transformations between them are the edges. The transformations estimated by the loop can be added to the graph as constraints in order to update the map [GKS+10]. It is possible, then, to correct the drift in the translation and rotation, but this does not solve the drift in the scale. Stasdat et al. [SMD10] proposed an optimisation based in seven degrees of freedom (DoF) constraints, unlike the traditional six DoF, to solve this problem. Results from this method can be seen on Figure 2.1.

2.2 Feature Matching

In order to detect the same point in different images it is necessary to match features among them. There are mainly two approaches to solve this problem [FS12]. One approach starts by detecting points of interest in the images and, posteriorly, describing those points in order to compare them

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figure 2.2: Comparison of feature detectors [FS12].

with points in other images. The other approach detects points of interest in one frame of a video and tracks them in the remaining frames.

Tuytelaars and Mikolajczyk [TM08] define a local feature (or interest point) as a pattern in the image which differs from its immediate neighbourhood. Points of interest should be:

1. *Repeatable:* A high percentage of the features detected on the scene that is visible in both images should be found in both images.

2. *Distinctive:* Features should contain enough information in order to be able to distinguish and match features on different images.

3. *Local:* The features should be local, so as to reduce the probability of occlusion.

4. *Numerous:* The number of feature points detected should be sufficient so it is possible to match features even on small objects.

5. *Accurate:* The detected features should be accurately localized, both in image location, as with respect to scale and shape.

6. *Efficient:* Preferably, the detection of features should be efficient enough to allow for time-critical applications.

Feature detectors try to find corners or blobs on images. A corner is defined as a point at an intersection of two or more edges, while a blob is a pattern on the image that differs from its neighbours in terms of intensity, texture and color. Several corner (such as Harris[HP87], Shi-Tomasi[ST94] and FAST[RD06]) and blob detectors (such as SIFT[Low04], SURF[BTG06] and CENSURE[AKB08]) were created over the years. Corner detectors are faster but less distinctive and repeatable, while blob detectors are more distinctive but slower to detect. A comparison of these detectors is presented on Figure 2.2.

Feature detectors can be divided in two steps. On the first step, a function is applied to the whole image. For example, Harris detector applies a corner response function while SIFT applies

the Difference of Gaussian (DoG). On the second step, all local minimums (or maximums) of the applied function are identified. To make the detector scale invariant, the function is applied to the image with different scales.

After the detection of features, it is necessary to describe them. One widely used descriptor is SIFT. This descriptor consists on a local histogram of the gradient orientations. The fragment around the feature is divided into a 4×4 grid and, for each quadrant, the histogram with the eight orientations of the gradient is computed. The 16 histograms of dimension 8 are concatenated resulting in a 128 dimension vector. In order to reduce the effects of illumination differences, this vector is normalized.

Recently, three new binary descriptors were developed that proved to be faster than SIFT and SURF. One descriptor called BRIEF [CLSF10] that, despite being extremely fast, is also distinctive in the absence of rotation and scale. Inspired by BRIEF, ORB [RRKB11] was developed that, beyond some optimizations in the point description, is invariant to rotation. At last, following the same lines, BRISK [LCS11] is based on FAST and is invariant to rotation and scale.

Bastos proposed the Fast Invariant to Rotation and Scale Transform (FIRST) [Bas09] algorithm in the context of Augmented Reality and Computer Vision. FIRST uses the Shi and Tomasi's method to detect corners and define features. The scale factor is determined by selecting the multiplication factor that maximizes the average of the luminance around the point. Similarly to SIFT, FIRST uses an histogram of the gradient in order to make the descriptor invariant to scale. Nonetheless, instead of saving the histogram, the image fragment is rotated by the maximum gradient and used as the descriptor.

Finally, it is necessary to match features detected on different images. The simplest way to do that is by comparing each feature descriptor of one image with all descriptors from the other image. Descriptors are compared using a similarity function. SIFT, for example, uses the Euclidean distance between the histograms that describe the points to perform the comparison.

Bastos [Bas09] compared FIRST with SURF and SIFT. The author concluded that SIFT is typically superior to FIRST and SURF in terms of repeatability and distinctiveness. Nonetheless, in the presence of noise and luminance variations, SURF obtained better results which suggests that the Determinant of Hessian (DoH) used by SURF is less sensitive to these perturbations than the DoG used by SIFT. The repeatability of FIRST is inferior to SIFT, however, it is similar to SURF. In terms of efficiency on the feature extraction and matching, SURF obtained better results than SIFT, wherein FIRST had the best efficiency. Finally, SIFT is the most exact and SURF the most prone to errors.

As stated previously, it is also possible to detect features in the first image and, posteriorly, track those points on the following images. This approach, known as the optical flow, performs well in video where the movement and deformation between adjacent frames is small. The optical flow is the apparent movement of brightness patterns on an image. Given the intensity values $I_t(p)$ e $I_{t+1}(p)$ at point $p = (x, y)$, where p is the central point of a region S, in two consecutive frames,

the movement of a region can be described by the following equation:

$$I_t(x,y) = I_{t+1}(x+d_x, y+d_y), \tag{2.2}$$

Where d represents the movement between two frames. This method works under some assumptions: the brightness is constant across all frames, the movement between frames is small, and the points move similarly to their neighbours.

One optical flow method that is widely used is the KanadeLucasTomasi (KLT) [LK81] where, instead of tracking all points, tracks only the corners.

A more recent tracking method, proposed by Li et al. [LWW12], uses SURF to extract features to be tracked. In order to perform the tracking over long sequences, the authors combine SURF with Incremental Visual Tracker (IVT). The method is capable of solving the drift problem when the object suffers a partial occlusion.

2.3 Point Cloud Reconstruction

Azernikov et al. [AMF03] defines the problem of surface reconstruction as a mesh M that approximates the surface of an unknown object O, given a dense point cloud P of object O, preserving its characteristics and topology. The authors classify the reconstruction methods in three categories: image processing, computational geometry and computer graphics approaches.

On the image processing approaches, the surface reconstruction problem can be seen as a special case of 3D space segmentation. These algorithms are based on active contours or "snakes" approaches [KWT88] wherein a contour of the surface is initialized and its position changes based on a predefined energy function.

Computational geometry approaches use hybrid functions in order to perform linear interpolation of unorganized points. This approach is based on combinatorial structures such as *Delaunay triangulations* [Boi84, KSO04], *alpha shapes* [EM94, BBX95, BMR+99] or *Voronoi diagrams* [ACK01, ABK98]. Mencl and Müller [MM98] proposed an algorithm based on graphs that approximates a wireframe to an unknown surface, in order to be filled with triangles.

Finally, computer graphics approaches have higher concerns with the visual quality of the resulting model. A widely known algorithm that fits in this category was proposed by Hoppe et al. [HDD+92]. A tangent plane is associated with all sub sets of points using Least Squares Fitting. After this step, the normals are extracted in order to construct a triangular iso-surface using the Marching Cubes [LC87] algorithm. Another method that fits in this category is the Poisson Surface Reconstruction and was proposed by Kazhdan, Bolitho and Hoppe [KBH06]. This method formulates the reconstruction of a surface as a Poisson problem. By doing that, the method uses global information creating smooth and robust to noise surfaces.

2.4 Distributed Algorithms

With the price reduction of image acquisition systems, the dimension of visual sensor networks has been increasing. For example, more than half a million cameras observe the streets of London. It is impossible to process this large amount of data in real time and, for that reason, there is the need to create distributed computer vision algorithms. Some work has been developed, mainly in the areas of distributed calibration of a camera network and in distributed tracking of one or more objects through the network.

Lee and Aghajan [LA06] used observations of a moving target obtaining its position using a distributed version of the Gauss-Newton method. Other similar method called Simultaneous Localization and Tracking (SLAT) [FGPS06], uses Gaussian densities and a process of linearization to cope with the uncertainty on the angles of the camera, in order to use the Kalman filter for the tracking.

Other algorithms need to know the intrinsic parameters of the cameras à priori, and use this information to estimate the rotation and translation between each pair of cameras. Barton-Sweeney et al. [BSLS06] used an extended framework of the Kalman filter on the estimated epipolar geometry, in order to determine the relative position between each pair of cameras.

There are methods that resort to the vision graph to share information. The method proposed by Devarajan et al. [DR04, DRC06] shares the projection points they have in common with their neighbours in the vision graph after some local processing, where the bundle adjustment is performed. Each camera estimates its own position and orientation as their neighbours'.

Roberto Tron et al. [TV11] presented how to implement distributed linear algebra algorithms, such as SVD and Least Squares, reducing them as distributed averages. With this knowledge it is possible to implement distributed versions of Machine Learning algorithms (PCA e GPCA) as well as Computer Vision algorithms (triangulation, pose estimation and affine SfM). However, these algorithms are far from being optimal and are used as an initialization for other non linear procedures, such as the bundle adjustment. Furthermore, on the communication point of view, these algorithms do not always bring big advantages. For example, the proposed distributed version of the SVD is not efficient when the rank of the data is of the same order as the number of samples.

Dopico et al. [DCSN04] proposed a parallel algorithm to compute the optical flow of a video sequence. The authors chose KLT as the optical flow algorithm, and they divided it in 5 tasks: temporal smoothing, spacial smoothing in x, spacial smoothing in y, computation of the partial derivatives and computation of the velocity of each pixel. The first four tasks are connected to a pipeline since they need data from several images to work correctly. The fourth task sends the partial derivatives of the complete images to different nodes on a rotative way. The authors used eight nodes: one for the first two tasks, other for the third and fourth tasks and the remaining six nodes for the last task, and also the most computationally expensive, in parallel. It is possible to use four, eight or sixteen nodes, as there are not sufficient tasks to distribute using more nodes. One way to overcome this limitation, increasing the level of parallelism, would be dividing the

image into several sub-images as proposed by Kohlberger et al. [KSBW03]. These sub-images would be independent provided that neighbouring sub-images had overlapping borders.

Dean and Ghemawat [DG08] from Google showed that it is possible to distribute a large number of real problems by using the map and reduce functions, present in most functional languages. The user needs to build a map function, that receives a key/value pair as input and produces a set of pairs intermediate_key/value $(k1, v1) \rightarrow \{(k2, v2)\}$. The MapReduce library aggregates all values associated with the same intermediate key and delivers them to the reduce function. The reduce function, also created by the user, receives an intermediate key and a set of values. This function aggregates all the values in order to create a smaller set of values than the input set. The authors state that, typically, each invocation of the reduce function returns zero or one value.

Mirashe and Kalyankar [MK10] define *cloud* as a large group of interconnected computers, publicly accessed via Internet. These computers can be personal, servers, public or private. The architecture and technologies used are transparent to the user.

The *cloud* architecture typically involves multiple components communicating through an Application Programming Interface (API) that, usually, correspond to web services. This architecture is extended to the client through a browser or a software application that access the *cloud* application. Data is distributed across several nodes, wherein each one delivers data to the client autonomously, avoiding central servers of metadata that can become bottlenecks.

The *cloud* has as advantages, among others, a greater compatibility among operating systems, automatic software updates, reductions in the costs of the infrastructure and on its maintenance, and increase of the computational power. However, there are some disadvantages such as the need for a constant Internet connection, the service may be slow and the data may not be secure.

Chapter 3

Robust Parallel Factorization

A method to perform Structure from Motion is required. As stated in Chapter 2, there are three main approaches to perform SfM. One of the approaches uses additional equipments and does not fit in the context of this thesis, since the reconstruction must be performed by using a monocular video. The factorization approach was chosen since it works directly with video, uses information from all frames, and it is usually faster than stereo vision approaches as it tracks points instead of matching them.

This Chapter is divided in 3 Sections. Section 3.1 reviews in depth the factorization method, Section 3.2 presents the method that was developed in the context of this thesis and Section 3.3 provides an extensive evaluation of the method.

3.1 Factorization

The factorization method was first presented by Tomasi and Kanade [TK92] in their breakthrough paper. Their method was briefly presented on Subsection 2.1.1 and is going to be detailed in this Section.

To the knowledge of the author, the factorization method and its subsequent extensions are not implemented in any library. Therefore, the original method, proposed by Tomasi and Kanade [TK92] was implemented. The perspective factorization [HK02] method, as it is more complex and would require more time, was not implemented and remains as future work. Nonetheless, the method presented in Section 3.2 is able to generalize for any existing offline factorization method.

3.1.1 Formalization

In video processing it is possible to obtain *feature tracks*, in other words, to detect a set of features in a video frame and, posteriorly, search their position in subsequent frames. This process was reviewed on Section 2.2. The process of factorization consists on using these *tracks* to obtain the

structure of an object that is present in the image sequence and movement of the camera. By using information from all frames, the method becomes more robust to noise.

Assuming that the position $x_{fp} = [u_{fp}, v_{fp}]^T$ of point p in frame f is known, and that the model of the camera is orthographic, implying that the last row of the projection matrix is always equal to $[0,0,0,1]$, it is possible to write the following relation:

$$x_{fp} = R_f s_p + t_f, \tag{3.1}$$

Where $R_f = [i_f, j_f]^T$ is a 2×3 orthonormal rotation matrix, s_p is the 3D position of point p and $t_f = [a_f, b_f]^T$ is the 2×1 translation vector for frame f. By writing all the horizontal feature coordinates u_{fp} into an $F \times P$ matrix U, where F is the number of frames and P is the number of points, and all the vertical feature coordinates v_{fp} into an $F \times P$ matrix V, the *measurement matrix* $W \in R^{2F \times P}$ can created:

$$W = \begin{bmatrix} U \\ \hline V \end{bmatrix} \tag{3.2}$$

The rows of W are registered or, in other words, translated to their centroid, by subtracting the mean of the entries in the same row:

$$\begin{aligned} \tilde{u}_{fp} &= u_{fp} - a_f \\ \tilde{v}_{fp} &= v_{fp} - b_f, \end{aligned} \tag{3.3}$$

Where

$$\begin{aligned} a_f &= \tfrac{1}{P} \sum_{p=1}^{P} u_{fp} \\ b_f &= \tfrac{1}{P} \sum_{p=1}^{P} v_{fp} \end{aligned} \tag{3.4}$$

The *registered measurement matrix* $\tilde{W}$ is created and is used as input for the factorization method. By registering the measures it is possible to write the following equations:

$$\begin{aligned} \tilde{u}_{fp} &= i_f^T s_p \\ \tilde{v}_{fp} &= j_f^T s_p \end{aligned} \tag{3.5}$$

Considering an orthographic projection, the camera orientation at frame f can be represented by orthonormal vectors i_f, j_f and k_f, where i_f corresponds to the x axis of the image plane, j_f to the y axis and $k_f = i_f \times j_f$. Therefore, the *registered measurement matrix* can be expressed in matrix form:

$$\tilde{W} = MS \tag{3.6}$$

It was proven by the authors that, in the absence of noise, the rank of matrix $\tilde{W}$ is at most 3, since it is the result of the product of the $2F \times 3$ matrix M and the $3 \times P$ matrix S.

3.1.2 Approximate Rank

By using SVD, $\tilde{W}$ can be factorized into two orthonormal matrices $U \in R^{2F\times r}$ and $V^T \in R^{r\times P}$ and a diagonal matrix $\Sigma \in R^{r\times r}$ with the non-zero singular values of $\tilde{W}$, in decreasing order, where r is the rank of matrix $\tilde{W}$:

$$\tilde{W} = U\Sigma V^T \tag{3.7}$$

In the presence of noise $r > 3$. To approximate the *registered measurement matrix* to rank 3, matrices $U' \in R^{2F\times 3}$, $\Sigma' \in R^{3\times 3}$ and $V'^T \in R^{3\times P}$ must be computed. The upper left 3×3 sub matrix of Σ yields Σ', U' is composed from the first three columns of U and V'^T from the first three rows of V^T. It is now possible to get the best rank 3 estimate of $\tilde{W}$ and of the structure and motion matrices by:

$$\hat{M} = U'(\Sigma')^{\frac{1}{2}} \tag{3.8}$$

$$\hat{S} = (\Sigma')^{\frac{1}{2}} V'^T \tag{3.9}$$

$$\hat{W} = \hat{M}\hat{S} \tag{3.10}$$

Nonetheless, the decomposition 3.10 is not unique, since any invertible 3×3 matrix Q can be used to produce matrices $\hat{M}Q$ and $Q^{-1}\hat{S}$, which are, also, valid decompositions of $\hat{W}$.

3.1.3 Metric Constraints

In fact, $\hat{M}$ is a linear transformation of the true motion matrix M and $\hat{S}$ a linear transformation of the true structure matrix S, such that:

$$M = \hat{M}Q \tag{3.11}$$

$$S = Q^{-1}\hat{S} \tag{3.12}$$

It is known that the rows of the true motion matrix M are unit vectors, and that the first F rows are orthogonal to the corresponding F rows in the second half of M:

$$i_f^T i_f = j_f^T j_f = 1 \text{ and } i_f^T j_f = 0 \tag{3.13}$$

Therefore, it is possible to apply these *metric constraints*, composing an overdetermined system of $3F$ equations such that:

$$\begin{aligned} \hat{i}_f^T L \hat{i}_f &= 1 \\ \hat{j}_f^T L \hat{j}_f &= 1 \\ \hat{i}_f^T L \hat{j}_f &= 0, \end{aligned} \tag{3.14}$$

Where $L \in R^{3\times3}$ is a symmetric matrix:

$$L = Q^T Q \tag{3.15}$$

Matrix Q can be retrieved by performing an eigendecomposition on $L = E\Lambda E^{-1}$. In practice, matrix L can be a non positive definite matrix, resulting on at least one eigenvalue being less or equal to zero. To find a positive definite approximation of L, all eigenvalues $\lambda_i \leq 0$ are set to a value $\varepsilon > 0$ and saved on matrix Λ', allowing the computation of Q:

$$Q = E\Lambda'^{\frac{1}{2}} \tag{3.16}$$

Finally, the true motion and structure matrices can be obtained. The reconstruction will be made up to an arbitrary scale and on an arbitrary pose. If desired, it is possible to align the first camera reference with the world reference system by applying a rotation $R_0 = [i, j, k]$ that rotates the first frame to the identity matrix. The motion and structure matrices can be updated by the products MR_0 and $R_0^T S$.

3.1.4 Occlusions

As the camera moves, feature points can appear and disappear from the image. As the video progresses, a point can be occluded or, if the video incurs in abrupt movement, the tracking can fail. In practice this results in a matrix W with some missing values. Tomasi and Kanade [TK92], in one of their experiments, obtained a matrix W with only 16% of known entries.

Nonetheless, since the method uses SVD, it can only perform the factorization when all the entries of $\tilde{W}$ are known. The authors overcame this problem by applying the factorization on a sub matrix of W, where all entries are known, and then, as they called it, *hallucinate* the unknown entries of W. The problem of finding the optimal sub matrix might not be a trivial one.

For an unknown image measurement (u_{fp}, v_{fp}) to be reconstructed, point p must be visible in at least three more frames f_1, f_2, f_3, and there must be at least three more points visible in all the four frames f_1, f_2, f_3,f.

Suppose that W is a 8×4 matrix with only one unknown image measurement u_{44} and v_{44}. Notice that the columns and rows of W can be permuted.

$$W = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \\ u_{31} & u_{32} & u_{33} & u_{34} \\ u_{41} & u_{42} & u_{43} & ? \\ v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & ? \end{bmatrix} \tag{3.17}$$

One can perform the factorization on the first three frames, that is, to the 6×4 sub matrix, or on the 8×3 sub matrix containing the first three points. Assuming the case where all frames are kept:

$$W = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \\ u_{41} & u_{42} & u_{43} \\ v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \\ v_{41} & v_{42} & v_{43} \end{bmatrix} \tag{3.18}$$

By applying the method to the 8×3 matrix W, the full translation t' and rotation M matrices and the partial structure matrix S' are produced:

$$W_{8\times3} = MS'_{3\times3} + t'e^T, \tag{3.19}$$

where $e = [1,1,1]$. Matrices with primes are represented relative to the centroid of only the first three points. By solving the following overconstrained system of six equations in three unknowns, point s'_4 can be computed:

$$\begin{bmatrix} i_1^T \\ i_2^T \\ i_3^T \\ j_1^T \\ j_2^T \\ j_3^T \end{bmatrix} s'_4 + \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} u'_{14} \\ u'_{24} \\ u'_{34} \\ v'_{14} \\ v'_{24} \\ v'_{34} \end{bmatrix}, \tag{3.20}$$

where

$$\begin{aligned} u'_{f4} &= u_{f4} - a'_4 \\ v'_{f4} &= v_{f4} - b'_4 \end{aligned} \; for\ f = 1,2,3 \tag{3.21}$$

The shape coordinates can, then, be registered with respect to their centroid:

$$s_p = s'_p - \frac{1}{4} S' e_4 \; for\ p = 1,2,3,4 \tag{3.22}$$

The translation can be found by equation 3.4. For the case where all points are kept and W is a 6×4 matrix, the centroid of the first three points is found so the motion of the remaining frame is found. Then, the translation vector is updated and the unknown entries are computed.

By combining this two *hallucination* approaches, it is possible to start with a small sub matrix and obtain the motion of the remaining frames and the position of new points.

In this thesis, the claim is made that over long video sequences, and in the presence of noisy measures, the *hallucination* method is bound to propagate the error. One of the reasons is that, since the system of Equation 3.20 is solved by least squares, one noisy measure will introduce an error that can propagate on further *hallucinations*. One way of alleviating this problem is by adding more equations to the system. If one point to be *hallucinated* is seen in more than three frames, it is possible to include that information in Equation 3.20. A better, but more expensive approach, would be to detect outliers and not using those to *hallucinate* new points.

As reviewed in Section 2.1.1, it is possible to use a matrix factorization algorithm that deals with missing values. These approaches can implicitly deal with noisy measures, and are preferred to the *hallucination* method, nonetheless none of these methods were implemented due to time constraints, and remains as future work.

3.2 Method

The previously described method runs sequentially. Nonetheless, in the context of this thesis, a distributable method is required in order to build a scalable *web service*. A method that is able to robustly perform SfM, even on long video sequences, is presented in this Section. The method deals with the problem of the propagation of the error, induced by the *hallucination* process, and is also able to detect outliers.

3.2.1 Technologies

Several libraries were used to ease the implementation of the method that was described on Section 3.1 and in this Section. Other computer programs were used in order to evaluate the method and to test different approaches and parameters, that were crucial for the success of the development and implementation of the method. The technologies that had the most impact are presented in this Subsection.

The most useful library was OpenCV[1], which is an open source computer vision library, with C, C++, Java and Python interfaces. It is written in optimized C/C++, and is able to take advantage of multi-core processing. Some features, such as the Mat class for matricial computations, SURF, KLT tracker, SVD and *k-d tree* were extremely helpful. In this project, the C++ interface was used.

Point Cloud Library (PCL[2]) is an open source library for 2D and 3D image and point cloud processing. It was used to display the generated point clouds and, since it implements point cloud reconstruction method such as the Marching Cubes [HDD+92] and Poisson Surface Reconstruction [KBH06], it was used for some experiments. The implementation of the Iterative Closest Point (ICP) was also helpful in the evaluation phase.

Eigen[3] is a C++ template library for linear algebra and was used since it implements the Umeyama's method [Ume91]. Also, it was used while interacting with PCL.

[1] http://opencv.org/
[2] http://pointclouds.org/
[3] http://eigen.tuxfamily.org/

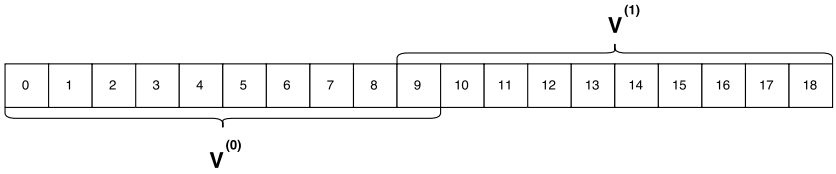


Figure 3.1: Video divided in two videos $v^{(i)}$ with one overlapping frame.

Blender[4] is another open source project. It is a free professional computer graphics software that is able to create animated films, visual effects, 3D models and video games. As such, it was used to create synthetic videos and obtain the ground truth position of the 3D points.

Finally, Meshlab[5] is a free open source mesh processing software. It was used for visual inspection of the point clouds, perform data cleaning and manual alignment of point clouds and, also, to test several parameters of the Poisson Surface Reconstruction and Marching Cubes.

3.2.2 Video Division

In order to distribute the method, a video can be divided into n videos $v^{(i)}$. Each video shares its last k frames with the next video such that the sequence of the frames is preserved. Then, each $v^{(i)}$ can apply the factorization method independently and, therefore, in parallel. A simple example is presented on Figure 3.1, where the video is divided in two videos $v^{(i)}$ of ten frames each, with one overlapping frame.

SURF features are extracted from the first frame of each video and, then, are tracked across the remaining frames using KLT. SURF was chosen since it has high repeatability and is scale invariant, which can be useful to match points on overlapping frames. KLT was chosen for its simplicity and since there is no need for more complex tracking methods, as the length of each $v^{(i)}$ will be relatively small.

The measures are saved on a $2F \times P$ matrix $W^{(i)}$, as described on subsection 2.1.1, and then passed as input for the factorization method, that outputs the structure of the scene and motion of the camera.

3.2.3 Registration

After the factorization step, there is a registration problem that needs to be solved. There are n point clouds, with overlapping points, and n rotation matrices and translation vectors with k overlapping frames.

In the absence of noise, one frame shared by two different videos must have the same pose. As the factorization method sets the centroid of the measures as the origin and with an arbitrary pose and scale, each video will be reconstructed under different coordinate systems. Nevertheless, all videos must share the same origin.

[4] https://www.blender.org/
[5] http://meshlab.sourceforge.net/

Another requirement is that the same point, reconstructed by different videos, must have the same 3D position. It is possible to perform point-matches by using the nearest neighbour approach between each pair $(v^{(i)}, v^{(i+1)})$. In order to do that, a *k-d tree* was built with the 2D points seen on the first frame that $v^{(i)}$ shares with $v^{(i+1)}$ and, for each 2D point seen on the first frame of $v^{(i+1)}$, the nearest point in the tree is found. All matches $(p^{(i)}, p^{(i+1)})$ with Euclidean distance less than ε are saved.

Having a set of matches, it is possible to compute a similarity transformation $A = [sR \quad t]$ that aligns $S^{(i+1)}$ with $S^{(i)}$, where $S^{(i)}$ is the structure of $v^{(i)}$. This problem can be solved using Umeyama's method [Ume91], which has a closed-form solution, and minimizes the mean squared error $\frac{1}{m}\sum_{p=0}^{m} ||S_p^{(i)} - (sRS^{(i+1)} + t)||^2$ between the set of m 3D matches $(S_p^{(i)}, S_p^{(i+1)})$. The two point clouds can, then, be aligned by:

$$S'^{(i+1)} = sRS^{(i+1)} + t \tag{3.23}$$

$$t'^{(i+1)} = st^{(i+1)} - M^{(i+1)}R^T t \tag{3.24}$$

$$M'^{(i+1)} = M^{(i+1)}R^T \tag{3.25}$$

$$W'^{(i+1)} = sW^{(i+1)}, \tag{3.26}$$

where $S'^{(i+1)}$, $t'^{(i+1)}$, $M'^{(i+1)}$ and $W'^{(i+1)}$ are the matrices of video $i+1$ aligned with video i. These equations guarantee the equality $W'^{(i+1)} = M'^{(i+1)}S'^{(i+1)} + t'^{(i+1)}$. Nonetheless, the minimization of the mean squared error might not be desirable in the presence of outliers. These outliers can appear due to errors in the tracking process or due to errors in the reconstruction process. To improve the robustness of the method, Random Sample Consensus (RANSAC) was used. Four points are randomly sampled and matrix A is computed. The RANSAC model is comprised of A and the mean squared error e between the four sample points, after the transformation A, and the corresponding matches in the other point cloud. All points that, by applying the transformation A, have a similar mean squared error to e are considered inliers.

As the computation of the similarity transformation degenerates when the 3D point sets form a plane, line or point [ELF97], it is important to guarantee that the four points used to obtain A are not coplanar. If you imagine that all four points have the same 3D position then, a simple translation could perfectly align the matches. However, there would be an infinite number of rotation matrices that would also perfectly align them. The same happens with points forming a line or plane as some information is lost.

Having four points A, B, C and D, it is possible to compute the vectors $\vec{AB}$ and $\vec{AC}$. By taking the cross product $\vec{N} = \vec{AB} \times \vec{AC}$, the normal vector to the plane that contains points A, B, and C is obtained. If the dot product between vector $\vec{AD}$ and $\vec{N}$ is 0, then all points are coplanar since the angle between $\vec{AD}$ and $\vec{N}$ is 90 degrees. In conclusion, four points are considered coplanar if the

result of the following equation is close enough to 0:

$$\vec{AD} \cdot (\vec{AB} \times \vec{AC}) \tag{3.27}$$

The four points are discarded if considered coplanar and another four are randomly selected until a set is found to be non coplanar.

3.2.4 Noise Reduction

The point match relation is transitive, meaning that if (p_1, p_2) and (p_2, p_3) are matches, then (p_1, p_3) are also matches. By using this property, it is possible to perform matches across multiple videos. In the presence of noise, the 3D positions of p_1, p_2 and p_3 might not be the same. To reduce the effects of the noise, an ideal point is computed for each point match by computing a weighted average of the points:

$$S^{(ideal)} = \frac{\sum_{p=0}^{m} r_p^2 S'_p}{\sum_{p=0}^{m} r_p^2} \tag{3.28}$$

Where r_p is the confidence in the reconstruction of the given point, and is computed by:

$$r_p = 1 - \frac{\sigma_4}{\sigma_3} \tag{3.29}$$

Where σ_i is the i^{th} largest singular value of the SVD factorization of the registered measurement matrix $\tilde{W}$. It was stated by Tomasi and Kanade [TK92] that, in the absence of noise, $\tilde{W}$ is at most of rank 3 and, since the number of non-zero singular values equals to the rank of a matrix, σ_4 should be equal to zero. By looking at equation 3.29, it is possible to conclude that the largest the value of r, the closer W is of rank 3.

Finally, it is possible to compute the 3×4 affine transformation $A^{(i)}$ that minimizes:

$$\sum_{p=0}^{m} ||S_p^{(ideal)} - A^{(i)} \tilde{S}_p'^{(i)}||^2, \tag{3.30}$$

Where $\tilde{S}_p'^{(i)}$ is the 3D point $S_p'^{(i)}$ in homogeneous coordinates. The final structure $S''^{(i)}$ of each video can then be computed by:

$$S''^{(i)} = A^{(i)} \tilde{S}'^{(i)} \tag{3.31}$$

By using $\tilde{S}_p'^{(i)}$, matrix A is able to translate $S'^{(i)}$. This allows the correction of potential errors on the registration process without the necessity to deform the point cloud. The noise reduction step can be done iteratively since there can be conflicting matches across several videos. The

criteria used to check for convergence is the variation of the error between point matches after applying the affine transformation.

$$\Delta^{(i)} = \left| \sum_{p=0}^{m} ||S_p^{(ideal)} - S'^{(i)}||^2 - \sum_{p=0}^{m} ||S_p^{(ideal)} - S''^{(i)}||^2 \right| \qquad \varepsilon = \sum_{i=0}^{n} \Delta^{(i)} \tag{3.32}$$

The system converges when ε is close enough to zero. Nonetheless, it is not guaranteed that the system will converge and there is the need for setting a maximum number of iterations.

The pose of the camera is not updated since the resulting matrix of $M^{(i)}A^{-1(i)}$ would not guarantee a valid rotation matrix.

3.2.5 Structure and Motion

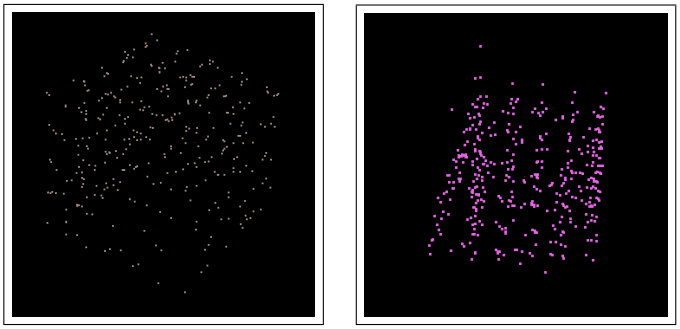
The results of all videos have to be concatenated into one unique solution. For that, the pose of the camera is vertically concatenated and the average is computed on overlapping frames. The average of the translation vector is straightforward, nonetheless, a simple average is not applicable to the rotation matrix. This problem can be solved, as reviewed by Claus Gramkow [Gra01], by finding the rotation matrix $\bar{R}$ that best rotates the identity matrix into the sum of the two rotations to be averaged. The solution can be obtained by SVD, $\sum_i R_i = U\Sigma V^T$, where R_i are the overlapping rotation matrices. Introducing the matrix $D = diag(1,1,det(U)det(V^T))$, the mean rotation is:

$$\bar{R} = UDV^T \tag{3.33}$$

Regarding the final structure, points fit into three categories: *Inliers*, *Outliers* and *Remaining points*, with the *Inliers* and *Outliers* being the result of the RANSAC step. For all matches in the inlier set, the ideal point is computed, as in equation 3.28, and added to the final structure. All points considered to be outliers are not added, since they can be caused by two situations: bad reconstruction or bad matching. The bad matching can, also, be caused by two situations: drift on the tracking process or the propagation of the error in the hallucination process. Both situations can indicate a bad reconstruction and, therefore, all outliers are ignored. Other points that were not matched between videos are also added to the final structure, since there is no indication of them being erroneous.

3.3 Evaluation

This Section presents an evaluation of the Robust Parallel Factorization Method (RPFM) proposed on Section 3.2. The method is evaluated qualitatively on Subsection 3.3.1, Subsection 3.3.2 evaluates the resulting structure and Subsection 3.3.3 the resulting motion. Subsection 3.3.4 provides an evaluation on the division of the video. Subsection 3.3.5 evaluates the noise reduction step and effects of varying the number of iterations. Subsection 3.3.6 compares the RPFM with 123D Catch



(a) (b)

Figure 3.2: Qualitative evaluation of resulting structure, by adding gaussian noise $\mathcal{N}(0,1)$ to a video sequence with 1275 frames. (a) RPFM structure. (b) Tomasi and Kanade's method structure.

on real videos and, finally, Subsection 3.3.7 evaluates the complexity of RPFM and compares it with the complexity of the classical method.

3.3.1 Qualitative Evaluation

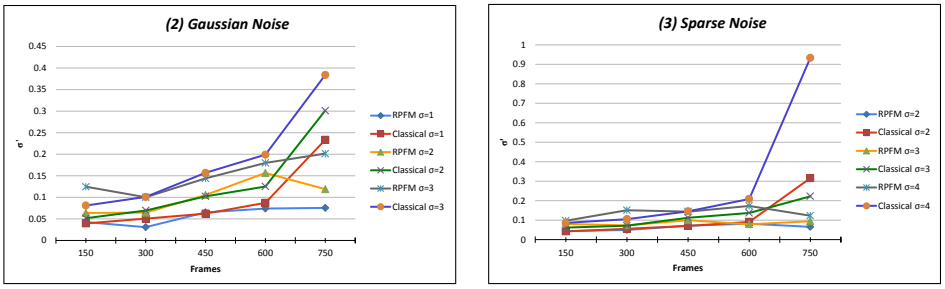
A synthetic cube with 98 points was created and a perspective camera was rotated around it, in a closed circle. To evaluate qualitatively the performance of both methods, Gaussian noise $\mathcal{N}(0,1)$ was added to a video sequence with 1275 frames, and the resulting point clouds are shown on figure 3.2. As can be seen, the point cloud generated by RPFM has less outliers and maintains the aspect of the cube, as the point cloud generated by Tomasi and Kanade's method created an inclined face.

Experiments showed that this is only true for long sequences, since smaller videos tend to be reconstructed with fewer points by RPFM, due to the outlier removal step, making it harder to understand the shape of the object the point cloud is representing.

3.3.2 Structure Evaluation

The method reconstructs the scene in an arbitrary scale, origin and rotation and, therefore, the evaluation of the structure is not a direct one. One possible approach would be to apply the Umeyama's method [Ume91] to align the reconstructed point cloud with the ground truth point cloud, and then obtain the RMSE between the two. There are some problems with this approach: the results would be influenced by the quality of the registration process, which might contain outliers, and the results would have to be normalized for proper comparison.

The chosen approach for the evaluation was to verify the consistency of the distances between points. The distance between points p_i and p_j of the ground truth structure S, and the distance between $\hat{p}_i$ and $\hat{p}_j$ of the structure to be evaluated $\hat{S}$ are computed and, then, the ratio $r_{ij} = \frac{dist(\hat{p}_i,\hat{p}_j)}{dist(p_i,p_j)}$



(a) (b)

Figure 3.3: Structure error with respect to the length of a video. (a) Added gaussian noise $\mathcal{N}(0,2)$ to the measures. (b) Added gaussian noise $\mathcal{N}(0,4)$ to 50% of the points.

is obtained, where $dist(p,p')$ computes the Euclidean distance between p and p'. This is done for all possible point pairs on the point cloud to be evaluated, obtaining the average ratio:

$$\bar{r} = \frac{\sum_{i=1}^{\hat{P}-1} \sum_{j=i+1}^{\hat{P}} r_{ij}}{\sum_{s=1}^{\hat{P}-1} s}, \tag{3.34}$$

with $\hat{P}$ being the number of points in structure $\hat{S}$. Intuitively, this is the average scale between the two point clouds. With this approach, the matches between points in S and $\hat{S}$ are needed and, therefore, a synthetic dataset is required.

Having $\bar{r}$ it is possible to compute the standard deviation σ. If the reconstruction has no error $\sigma = 0$. A low σ means that the majority of point pairs agree on the scale and, therefore, the error is low. For this approach to work with different scales, a normalization step is required. To do that, the quotient between the standard deviation and the average is used to evaluate the performance of the system:

$$\sigma' = \frac{\sigma}{\bar{r}} \tag{3.35}$$

Noise was added to the measures and these were used as input for RPFM and for Tomasi and Kanade's method. Three types of noise were added: (1) *No noise*, (2) *Gaussian noise*, (3) *Sparse Noise*: 50% of the points were corrupted with Gaussian noise. The performance was evaluated by varying the amount of noise and the length of the video sequence. The results are presented in Table 3.1.

With the factorization method, it is very important to find a good sub matrix W to apply the SVD and obtain good structure and motion matrices. If W is too noisy, the error is going to propagate in the *hallucination* step, since new points are going to be computed from previously *hallucinated* points. That can be seen on Figure 3.3, where the error grows with the length of the video.

With RPFM, the negative effects of bad reconstructions are attenuated by the good reconstructions, reducing the error with respect to the length of the video. Where this method performs best

Frames	150	300	450	600	750
(1) No Noise					
RPFM	0,0252	**0,0255**	0,0385	0,0460	**0,0435**
Classical	**0,0214**	0,0289	**0,0322**	**0,0355**	0,0565
(2) Gaussian Noise					
RPFM $\mathcal{N}(0,1)$	0,0426	**0,0308**	0,0651	**0,0740**	**0,0755**
Classical $\mathcal{N}(0,1)$	**0,0395**	0,0506	**0,0622**	0,0873	0,233
RPFM $\mathcal{N}(0,2)$	0,0637	**0,0646**	0,105	0,157	**0,119**
Classical $\mathcal{N}(0,2)$	**0,0517**	0,0695	**0,102**	**0,125**	0,301
RPFM $\mathcal{N}(0,3)$	0,125	**0,100**	**0,144**	**0,180**	**0,202**
Classical $\mathcal{N}(0,3)$	**0,0810**	0,101	0,157	0,199	0,384
(3) Sparse Noise					
RPFM $\mathcal{N}(0,2)$	0,0439	**0,0496**	0,0717	**0,0817**	**0,0655**
Classical $\mathcal{N}(0,2)$	**0,0422**	0,0555	**0,0704**	0,0908	0,316
RPFM $\mathcal{N}(0,3)$	0,0802	0,0732	**0,0998**	**0,0789**	**0,0945**
Classical $\mathcal{N}(0,3)$	**0,0616**	**0,0717**	0,112	0,137	0,222
RPFM $\mathcal{N}(0,4)$	0,0972	0,151	**0,144**	**0,173**	**0,123**
Classical $\mathcal{N}(0,4)$	**0,0862**	**0,105**	0,145	0,209	0,933

Table 3.1: Structure error σ' of RPFM and the Tomasi and Kanade's classical method, under different noise conditions.

is in the case of *Sparse Noise* since it is able to detect the outliers and remove them from the final structure. This characteristic might be desirable, for example, on cases of apparent motion due to illumination changes. From Figure 3.3 it is possible to conclude that, in the case of *Sparse Noise*, the error of the RPFM maintains almost constant and, on the other hand, the error of the classical method starts to increase after 600 frames.

3.3.3 Motion Evaluation

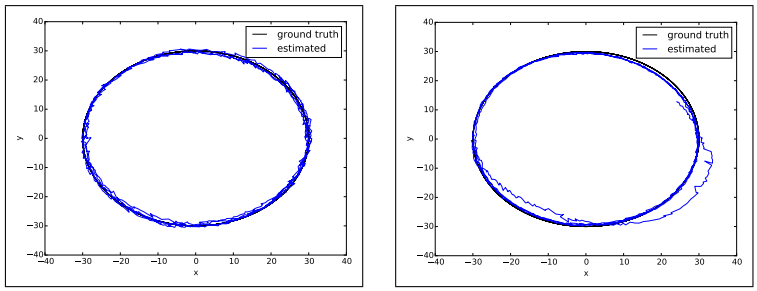
To evaluate the motion, code provided by Sturm et al. [SEE+12] was used to compute the Absolute Translation Error (ATE) and Relative Pose Error (RPE). ATE directly measures the difference between points of the true and estimated trajectory. The two poses are aligned using SVD and the difference between each pair of poses is computed. RPE is used to measure the drift of the system. The error in relative motion between all pairs of timestamps is computed.

The ATE was computed over the 3 types of noise mentioned previously and are presented in Table 3.2. Tomasi and Kanade's algorithm outperformed the RPFM on the *No Noise* case, probably due to errors in the registration process. Nonetheless, RPFM deals with the increase of noise better than Tomasi and Kanade's method. A comparison of the estimated trajectories is presented in Figure 3.4.

The same strategy was used to compute the RPE and the results are presented in Table 3.3. Once again, RPFM only outperformed Tomasi and Kanade's algorithm when noise was added to the measures. The median error is similar on both methods since Tomasi and Kanade's method

	(1) No Noise		*(2) Gaussian Noise* $\mathcal{N}(0,1)$		*(3) Sparse Noise* $\mathcal{N}(0,2)$	
	RPFM	Classical	RPFM	Classical	RPFM	Classical
median	1,52	**0,455**	**1,84**	2.02	**2,33**	3,19
max	3,59	**1,45**	**5,21**	9,92	**6,89**	17,1
RMSE	1,89	**0,551**	**2,04**	3.34	**2,83**	5,60

Table 3.2: Comparison of the Absolute Translation Error of the RPFM and Tomasi and Kanade's method using a video with 900 frames. The best results are shown in bold.



(a) (b)

Figure 3.4: Absolute Trajectory Error. The estimated trajectory, computed by adding Gaussian noise $\mathcal{N}(0,1)$ to the measures, is compared with the ground truth. (a) RPMF. (b) Classical algorithm.

	(1) No Noise		*(2) Gaussian Noise* $\mathcal{N}(0,1)$		*(3) Sparse Noise* $\mathcal{N}(0,2)$	
	RPFM	Classical	RPFM	Classical	RPFM	Classical
median	**0,0720**	0,0813	**0,0857**	0,136	**0,0971**	0,101
max	13,5	**11,7**	**15,7**	44,9	**14,6**	28,8
RMSE	5,10	**5,06**	**5,61**	13,3	**5,99**	9,05

Table 3.3: Comparison of the Rotational error, in degrees, of the RPFM and the Tomasi and Kanade's method using a video with 900 frames. The best results are shown in bold.

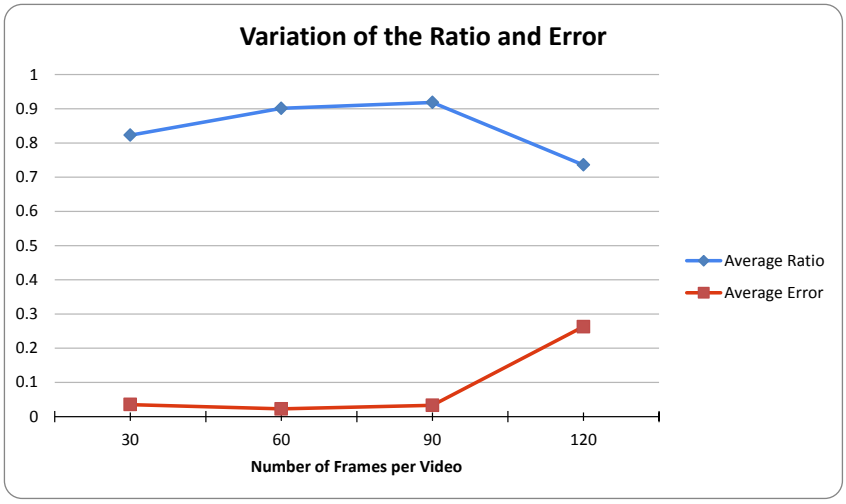


Figure 3.5: Variation of the average error and ratio between the fourth and third singular values with respect to the number of frames per video.

only starts getting worse estimates as the length of the video increases, due to the propagation of the error. That can be seen by the increasing RMSE and maximum error.

3.3.4 Video Division Evaluation

Up until now, the video was divided by defining an equal number of frames for each $v^{(i)}$. This was not a problem since the video was synthetic with a constant velocity and movement. The optimal number of frames depends mainly on the movement of the camera and, therefore, it might be different for each sequence and even for each video $v^{(i)}$.

In this subsection the effects of varying the number of frames for each video and the number of overlapping frames are evaluated. The same synthetic video is used and the reconstruction was performed with 5 videos $v^{(i)}$. For that, the videos were set to 30, 60, 90 and 120 frames. Also, for each number of frames, the number of overlapping frames was set to 1, $\frac{50}{3}\%$, 25% and 50% of the number of frames.

The ratios between the fourth and third singular values vary mostly with the number of frames per video and not with the number of overlapping frames. The average of the errors and ratios was computed for each maximum number of frames and the results are presented on Figure 3.5.

It is possible to conclude that the confidence increases with the number of frames up to a point, where the confidence decreases and the error increases. This happens since the number of points that are successfully tracked across all frames start to decrease, and most of the reconstructed points are *hallucinated*. Also, since less points are factorized, the factorization is more sensitive to noise.

The number of overlapping frames impacts the number of matches between points. With more overlapping frames, there is less error on the tracking process and more points are visible. This assumption is best tested on a real sequence, where the points are tracked using the KLT tracker.

	30	60	90	120
1	2400	1322	1009	769
$\frac{50}{3}\%$	2566 (6.9%)	1584 (19.8%)	1075 (6.5%)	938 (22.0%)
25%	2866 (19.4%)	1901 (43.8%)	1298 (28.6%)	991 (28.9%)
50%	3160 (31.7%)	2099 (58.8%)	1668 (65.3%)	1296 (68.5%)

Table 3.4: The absolute number of matches and the relative increase of matches, on the "medusa" video, with different number of frames and overlapping frames.

The "medusa"[6] video sequence was used and the number of matches was found with different number of frames and overlapping frames. The results can be seen on Table 3.4.

It is possible to conclude that the number of matches increases with the number of overlapping frames and decreases with the number of frames. As the number of frames increases more points are lost on the tracking process and, also, the existing points are more prone to drift.

The absolute number of matches is important but it does not make clear the relative gain, with respect to the number of matches, of adding more overlapping frames. In order to overcome that problem, the percentual increase of matches, relative to the number obtained by simply overlapping one frame, was computed and is also presented on Table 3.4.

It is possible to conclude that the gain is not deterministic and will depend on the video sequences. For example, videos with 30 or 90 frames have a gain of approximately 7% of matches by overlapping $\frac{50}{3}\%$ of frames and, on the other hand, videos with 60 or 120 frames gain approximately 20%. In most cases, the relative increase of matches is greater than the increase of overlapping frames. More matches may imply a better alignment and outlier detection, but the increase of overlapping frames increases the complexity of the method.

3.3.5 Noise Reduction Evaluation

The claim is made that, by performing the noise reduction step described on Subsection 3.2.4, good reconstructions improve the results of bad reconstructions. The noise reduction step is evaluated in this Subsection, on synthetic data and on the widely known "medusa" video.

Each video $v^{(i)}$ was limited to 50 frames, each one with 15 overlapping frames and the reconstruction was performed using the first five videos. The impact of varying the number of noise reduction iterations was evaluated.

The reconstruction was first performed on synthetic data with Gaussian noise $\mathcal{N}(0,1)$ added to the measures, by varying the number of noise reduction iterations and comparing the results with the ground truth. The results are presented on Table 3.5. The errors of the reconstruction are all similar, up to a hundred iterations and, for a thousand iterations the error increases.

By looking to the ratios between the fourth and third singular values of the synthetic video, it is possible to conclude that they are all similar. All reconstructions have similar quality and, therefore, the overall structure can not be improved. Nonetheless, for larger number of iterations,

[6] http://www.cs.unc.edu/~marc/

Iterations	0	5	10	20	100	1000
Error (σ')	0.113	0.109	0.112	0.113	0.112	0.147

Table 3.5: σ' error of the reconstruction, by varying the number of noise reduction iterations, on synthetic data with $\mathcal{N}(0,1)$ Gaussian noise added to the measures.

the results seem to become worse. As stated on Subsection 3.2.4, the system is not guaranteed to converge. In practice, what might happen is that most points start to converge to the solution and, when they converge, the fewer noisier points, as they are more distant to their ideal point, get more weight on the solution and start to introduce error.

If there is an imbalance on the system, where one point or group of points starts to pull the other, the system will eventually converge. By defining the threshold on the ε of Equation 3.32 to be 0.0001, the system converged after around 17500 iterations into a form closer to a plane. For a threshold of 0.00001 the system converged in around 33000 iterations into a line and, for 0.000001 the system took around 49000 iterations to converge into a smaller line, closer to a point (Figure 3.6).

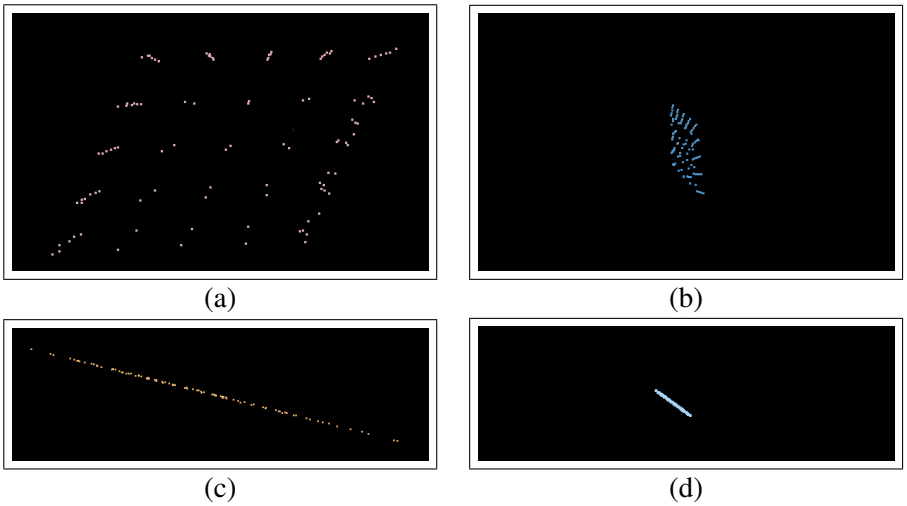
Nonetheless, Table 3.6 shows that the ratios of the videos taken from the "medusa" video sequence are more diverse and, therefore, more suitable to test the assumption of good reconstructions improving the results of bad reconstructions. As can be seen on Figure 3.7(a), one of the reconstructions is noisy. The points located on the lower left corner of the image should be closer to the big "mass" of points on the center. Nonetheless, by applying ten iterations of noise reduction (Figure 3.7(b)), the points are closer to their "real" position and after twenty iterations (Figure 3.7(c)) the reconstruction becomes perfectly aligned. The weight of the four well reconstructed point clouds was enough to reduce the noise of one badly reconstructed point cloud.

However, if the number of iterations is too high, the reconstruction starts loosing detail. It is not very visible on Figure 3.7(d), but after 100 iterations the resulting point cloud is almost flat. This can be seen on Figure 3.8(b), where almost all details of the image have disappeared. The face of the medusa, which is located on the center of the figure, is almost unrecognisable. On the other hand, by letting the system converge with a threshold of 0.1, in 13 iterations, the resulting model preserves some details. The face of the medusa is visible and even some texture of the wall and of the hair is visible.

It is advisable to limit the number of iterations and, also, set a higher threshold for the convergence test of Equation 3.32. Empirical tests showed that typically setting the maximum number of iterations to 20 and the threshold to 0.01% of the dimensions of the point cloud offers good

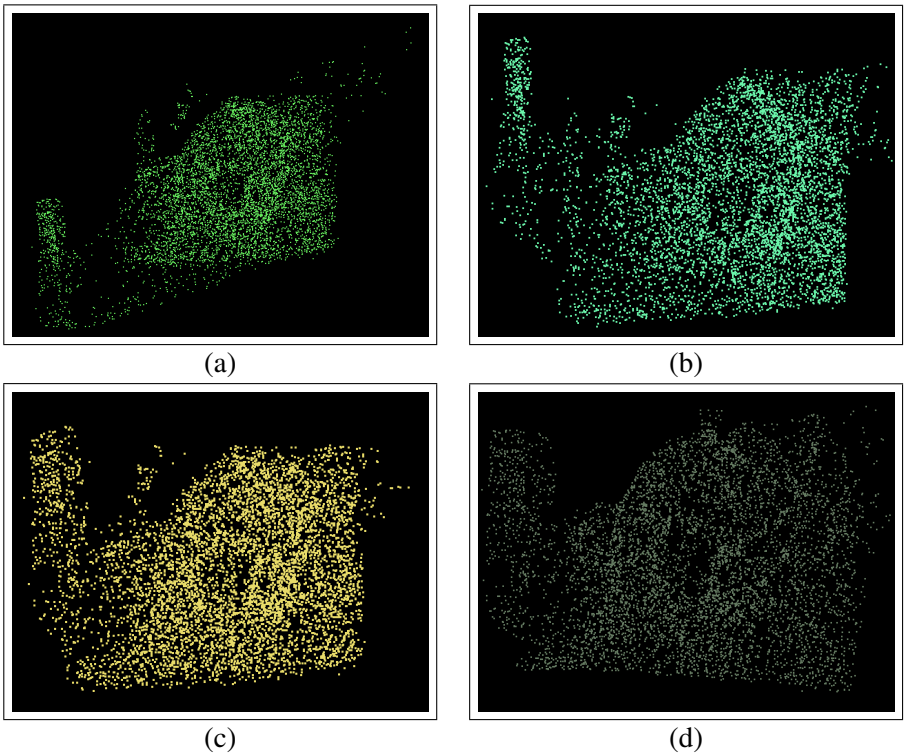
Video	1	2	3	4	5
Synthetic Ratios	0.910	0.857	0.871	0.916	0.901
Medusa Ratios	0.628	0.609	0.528	0.663	0.879

Table 3.6: Ratios r of the first five videos $v^{(i)}$ obtained from a synthetic and the "medusa" videos.



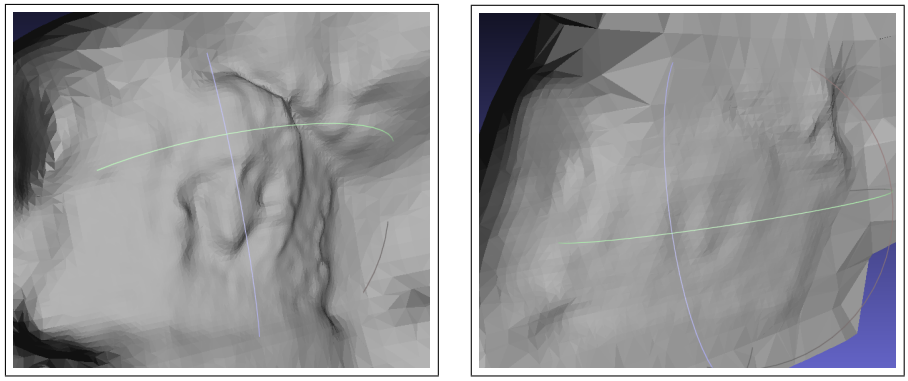
(a) (b) (c) (d)

Figure 3.6: Reconstruction of synthetic cube with different noise reduction iterations. (a) 2000 iterations. (b) 17000 iterations. (c) 33000 iterations. (d) 49000 iterations.



(a) (b) (c) (d)

Figure 3.7: Results of varying the number of iterations of noise reduction. (a) 0 iterations. (b) 10 iterations. (c) 20 iterations. (d) 100 iterations.



(a) (b)

Figure 3.8: Reconstructed models of the "medusa" video using the Poisson Surface Reconstruction method. (a) Point cloud after convergence in 13 iterations with threshold equal to 0.1. (b) Point cloud from Figure 3.7(d).

results. Nonetheless, the function used to check for convergence is not the best and should take into account the structure of the point cloud.

3.3.6 Comparison with 123D Catch

Autodesk 123D Catch is a mobile application that allows the user to take a series of photographs of a scene and reconstructs it using a *web service*. The application uses the *smartphone*'s sensors, such as the accelerometer and gyroscope, in order to obtain the relative pose between photos.

Chandler and Fryer [CF13] conducted an experiment to assess the accuracy of the application. They used a reflectorless total station to derive some control points and compare those with the points obtained by the application. For that, they applied a similarity transformation between the two point clouds and the residuals were computed. The overall standard deviations of the distance between the points are 12, 11 and 4 mm in XYZ respectively. The authors state that the accuracy is comparatively low to normal stereo close-range photogrammetry, but the task is fully automated and, therefore, simpler.

It is likely that, since the experiment of Chandler and Fryer, the accuracy of the application has increased. Also, the information from the *smartphone*'s sensors was not used by the authors. It makes sense to compare the results of 123D Catch with those of the RPFM, since both do not require previous camera calibration and are aimed for the *web*.

In order to compare both applications, a reconstruction of the same scene is obtained using the two approaches. The object that was chosen to be reconstructed was a Lego®'s Big Ben. Twenty four photos were taken around the object and sent for the 123D catch to reconstruct. It took around 1h02m30s for the application to finish the reconstruction with 206.345 vertices. As can be seen on Figure 3.9, the final reconstruction is not perfect, mainly on untextured areas.

As a preprocessing step, for better evaluation, the 123D Catch's model was scaled to real world measures. The distance d between two points in the real object was measured in *cm* and,

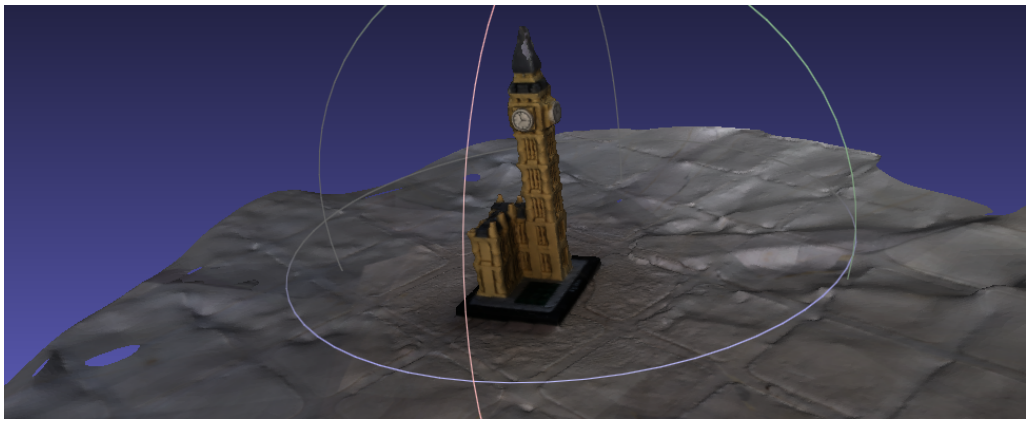


Figure 3.9: Textured model built by the 123D Catch application.

the distance $\hat{d}$ between the same points in the model was also measured. Then, the scale between the model and the real world can be obtained by:

$$scale = \frac{d}{\hat{d}} \tag{3.36}$$

By multiplying all points of the model by *scale* the distance between all points is measured in *cm*.

To perform the reconstruction with the RPFM, a video of the same scene as the one reconstructed by 123D Catch was taken using the camera of an *iPhone 4*. The video can be accessed on `http://tinyurl.com/q32497b` and one frame of the video can be seen on Figure 3.10. The whole video has 1012 frames and was filmed outdoors without any special care with respect to the light conditions and the movement of the camera. It is possible to see that the camera shakes as the user walks around the object. The camera moves in a circle around the object so it can be seen from all sides. Also, the object has some untextured areas which hampers the extraction and tracking of features. This is, therefore, a realistic and challenging test.

The video was reconstructed by using 7 videos $v^{(i)}$ with 150 frames and 25 overlapping frames. The resulting point cloud has 4549 vertices and converged in 5 iterations. The reconstruction took 50 seconds to finish on a computer with an Intel® Core™i7-2670QM with 2.20GHz and 4 cores. It could take less time if it used the GPU.

A comparison of the two point clouds is presented on Figure 3.11. It is possible to see that the cloud reconstructed by the 123D catch contains more points and, therefore, has more detail. Nonetheless, the RPFM cloud was able to extract points even on areas with no texture, such as the top of the object.

In order to compare quantitatively the two reconstructions, a variation of the ICP was implemented. A similarity transformation was computed, using the Umeyama's method [Ume91], instead of a simple rigid body transformation. This was needed since the two clouds are reconstructed with different scales.

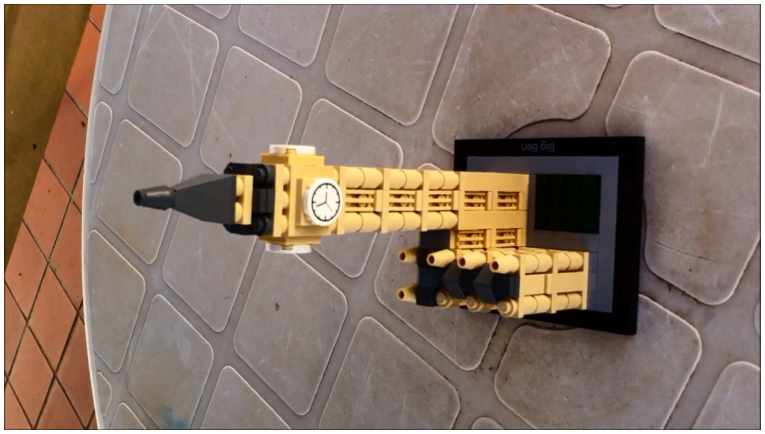
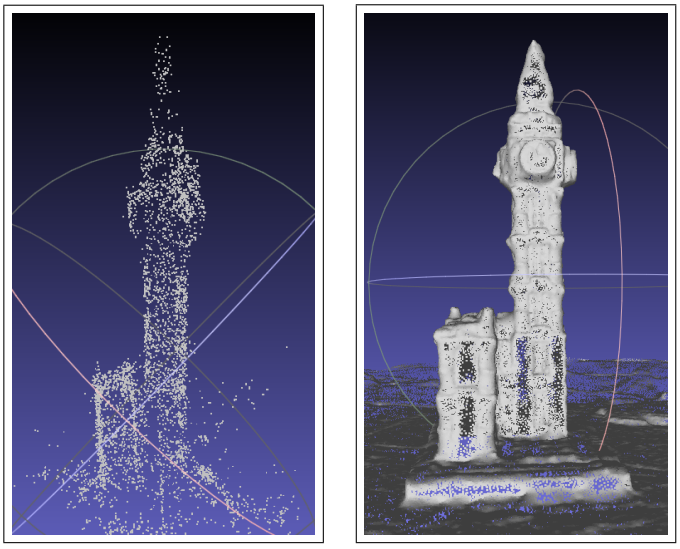
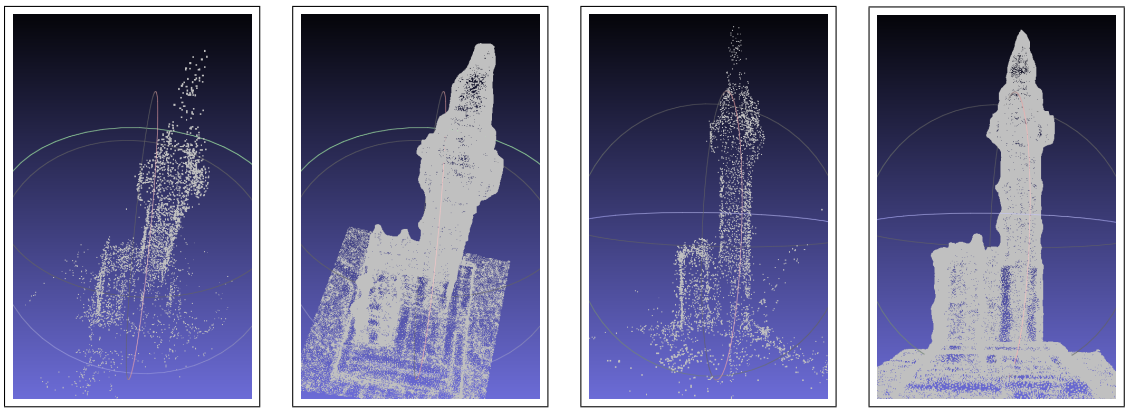


Figure 3.10: One frame of the video used to compare the RPFM with 123D Catch and a QRcode to the video that is located on http://tinyurl.com/q32497b.



(a) (b)

Figure 3.11: Qualitative comparison of the point clouds reconstructed by the 123D Catch application and RPFM. (a) RPFM. (b) 123D Catch.



(a) (b) (c) (d)

Figure 3.12: The aligned point clouds viewed from the same perspective. (a) RPFM. (b) 123D Catch. (c) RPFM. (d) 123D catch.

The point clouds built by RPFM and 123D Catch were manually cleaned and aligned to improve the results of the registration process. Since the two clouds are created with a large scale difference (RPFM cloud was 50 times bigger than the 123D catch's), the manual alignment is important in order to obtain good results.

Finally, the automatic alignment is performed. The sum of the squared distances between the computed point correspondences was $0.424cm$. The two aligned point clouds can be viewed in the same perspective in Figure 3.12. It is possible to confirm that the two point clouds have approximately the same height.

However, the houses of the parliament near the clock tower are relatively smaller in the RPFM's cloud than in the 123D Catch's one. This might happen since the factorization method that is being used assumes an orthographic projection and it does not preserve the scale between the different elements in the video. As the tower is closer to the camera than the houses, it is reconstructed as being relatively bigger. This effect can be seen in Figure 3.13. The distance between the real 3D points S_1 and S_2 is the same as the distance between S_3 and S_4. As these points are projected on the image plane, the assumption of an orthographic projection reconstructs the points as S'_1, S'_2, S'_3 and S'_4. As S_1 and S_2 are more distant to the camera than S_3 and S_4, they are projected closer in image coordinates. Therefore, the distance between S'_1 and S'_2 is smaller than the distance between S'_3 and S'_4. The aforementioned problem can be solved by implementing the perspective factorization created by Han and Kanade [HK02].

Taking into account that the system assumes an orthographic projection and that the video was a challenging one, the results were very good. By looking at the point cloud, there are no discontinuities or noise that could arise on the registration of the several independent reconstructions.

Another object was used for comparing the two approaches. This time, a rubik's cube was used. The 123D Catch application struggled with the reconstruction of the cube. Since the object lacks of distinctive features it is hard to match points between images due to the correspondence problem. Two attempts were made to reconstruct the cube with the 123D Catch application that

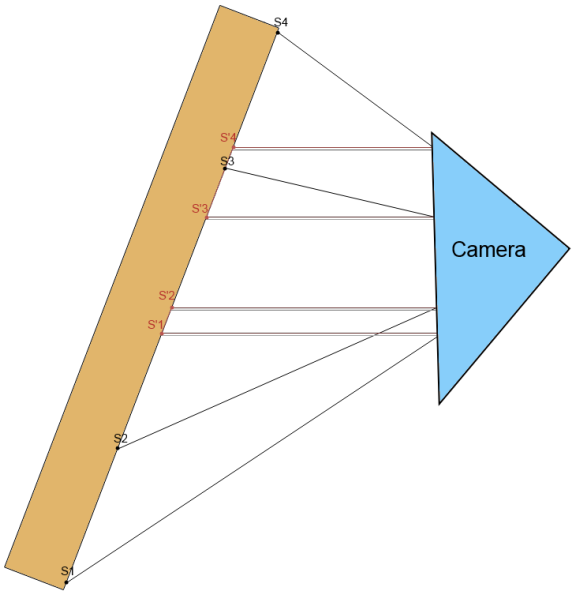


Figure 3.13: Explanation for scale difference by assuming an orthographic projection.

are presented in Figure 3.14. The first attempt was only able to reconstruct one face and the second three faces.

The video that was used for the reconstruction of the cube using the RPFM is available at `http://tinyurl.com/nc2vmjh` and one frame of the video can be seen on Figure 3.15. The video was also taken with the camera of an *iPhone 4*, with no special care with the movement of the camera nor with the light conditions. In fact, some reflections are seen on the top face of the cube throughout the video.

Since RPFM tracks points instead of matching points across frames, it was able to perform the reconstruction of the cube, and the results are presented on Figure 3.16. The lack of distinctive feature points also affects the quality of the tracking process and, therefore, the results have some noise. Nonetheless, RPFM was able to reconstruct all the faces of the cube and, in this case, performed better than the 123D Catch.

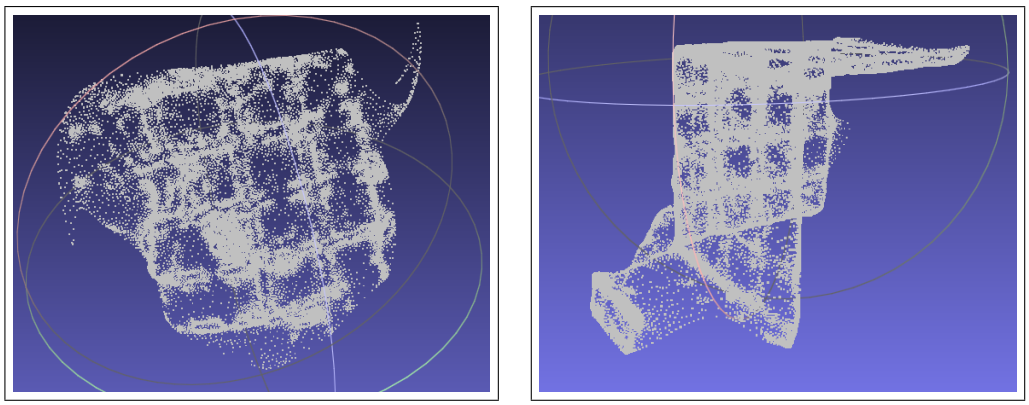


Figure 3.14: Two attempts to reconstruct a rubik's cube using 123D Catch.

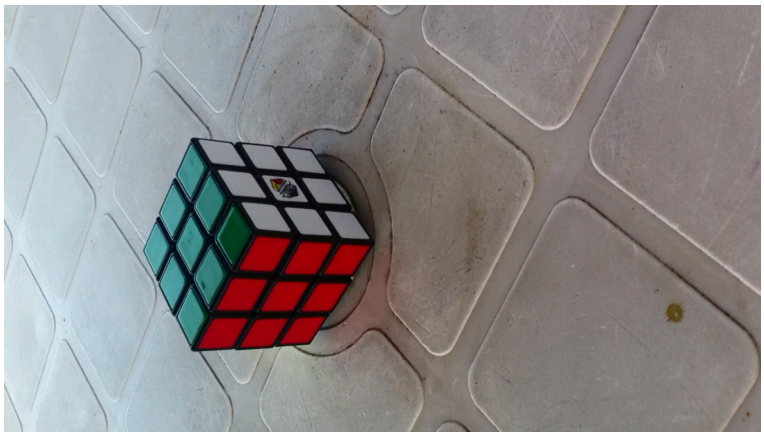


Figure 3.15: One frame of the video used to reconstruct the rubik's cube and a QRcode to the video that is located on http://tinyurl.com/nc2vmjh.

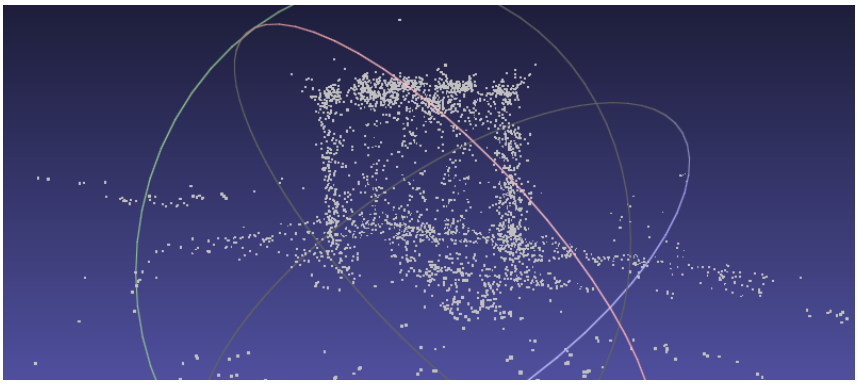


Figure 3.16: Rubik's cube reconstruction using the RPFM.

3.3.7 Complexity Evaluation

The factorization method uses SVD, which has a cubic complexity $\mathcal{O}(n^3)$ for square matrices, with n being the number of columns/rows. For matrices that are not square, the complexity becomes $\mathcal{O}(mn^2)$ with $n = min(rows, columns)$ and $m = max(rows, columns)$. The complexity of the classical algorithm is, excluding the tracking process and assuming that all points are visible on all frames, the same as the complexity of the SVD applied to the measurement matrix. Defining F as the number of frames and P as the number of points and, if $F \geq P$ then, the complexity equals to $\mathcal{O}(FP^2)$ otherwise, and in the more common case of $P \geq F$, the complexity of the factorization equals to $\mathcal{O}(PF^2)$.

By dividing the video in n videos $v^{(i)}$ with k overlapping frames, the number of frames for each $v^{(i)}$ becomes $f = \frac{F+k(n-1)}{n}$. In the worst case, where P stays constant, the complexity of each factorization becomes $\mathcal{O}(Pf^2)$, assuming that there are n cores or nodes to perform the computation in parallel. In the cases that $P < f$, and note that f is going to be a small number, the complexity becomes $\mathcal{O}(fP^2)$.

Nonetheless, RPFM has more steps to perform than the classical method. The noise reduction step starts by pairing all consecutive videos $(v^{(i)}, v^{(i+1)})$ in order to find point matches between them. As a simplification, it is assumed that all videos have the same number of points P. The complexity of building a *k-d tree* is $\mathcal{O}(P \log P)$ and the search for the nearest neighbour of P points is $\mathcal{O}(P \log P)$. As such, for each pair of videos, the matching takes $\mathcal{O}(P^2 \log^2 P)$.

In order to align the two videos the RANSAC method is applied. At each iteration, RANSAC computes the transformation between the matches of the two videos. In the worst case, the number of matches is the same as the number of points and the RANSAC performs the maximum number of iterations max_iter. In such scenario, the complexity is $\mathcal{O}(max\_iter(P-4))$. Finally, the transitive relations are found that, in the worst case take $\mathcal{O}(nP)$ time, and the affine transformations are computed in $\mathcal{O}(max\_iter)$ time. In practice, the number of iterations for the noise reduction process can be different from the number of RANSAC iterations however, they are represented by the same variable for simplicity.

Finally, we need to concatenate all the reconstructions. For k overlapping frames, the complexity of merging all poses is $\mathcal{O}(k(n-1))$. Since it is necessary to compute the ideal point for each *inlier*, the complexity of merging the point clouds is $\mathcal{O}(P)$.

The final complexity of the method becomes:

$$\begin{aligned} &\mathcal{O}(Pf^2 + P^2 \log^2 P + max\_iter(P-4) + nP + max\_iter + k(n-1) + P) \\ &\quad = \mathcal{O}(P[f^2 + P \log^2 P + max\_iter + n + 1] - 3max\_iter + k(n-1)), \end{aligned} \tag{3.37}$$

When $P \geq f$. Otherwise, when $f \geq P$, the complexity becomes:

$$\begin{aligned}&\mathcal{O}(fP^2 + P^2\log^2 P + max\_iter(P-4) + nP + max\_iter + k(n-1) + P)\\&= \mathcal{O}(P[fP + P\log^2 P + max\_iter + n + 1] - 3max\_iter + k(n-1))\end{aligned} \tag{3.38}$$

One can conclude that this method is computationally expensive even when performed in parallel. There are three possible variable combinations that need to be evaluated in order to make a proper comparison with the original method:

$$\begin{aligned}P \geq F \geq f\\F \geq P \geq f\\F \geq f \geq P\end{aligned} \tag{3.39}$$

A comparison of the complexities is going to be conducted. For that, there are some constraints that need to be enforced in the system:

$$\begin{aligned}max\_iter \geq 1\\n \geq 2\\P \geq 4\\f \geq 4\\f > k \geq 1\end{aligned} \tag{3.40}$$

The formulas for the complexity only apply for $n \geq 2$, otherwise it would work just as the classical method. Also, the number of iterations has to be greater than one in order to align the point clouds. In order to perform the reconstruction, there must be at least four points seen in four frames. Finally, it does not make sense for the number of overlapping frames to be equal or greater than the number of frames of each $v^{(i)}$.

The impact of changing n, k and max_iter was evaluated, when $P \geq F \geq f$. By setting $k = 1$, $n = 5$ and $max\_iter = 1$, it was possible to confirm that RPFM was able to performed better than the classical method on some situations, as it is presented on Figure 3.17(a). For the valid region of this case, after imposing all the constraints, the classical method performs better only on the space below the blue curve and above the purple one. It is possible to conclude that in this situation, RPFM performs better when the number of frames is larger.

By increasing the number of overlapping frames, there is the need for more frames for the RPFM to perform better (Figure 3.17(b)). The system is more sensible to the variation of overlapping frames than the number of iterations. Even after setting the maximum number of iterations to 120, the area where RPFM performs better is almost unchanged (Figure 3.17(c)). With the increase in the number of videos, there is the need for more frames, as each video requires at least 4 frames to perform the reconstruction. By analysing Figure 3.17(d), it is possible to conclude that the system does not improve by adding more videos in this situation.

The same analysis was performed for the case where $F \geq P \geq f$, with the same variation on n, k and max_iter, and presented in Figure 3.18. In this situation, by setting $k = 1$, $n = 5$ and

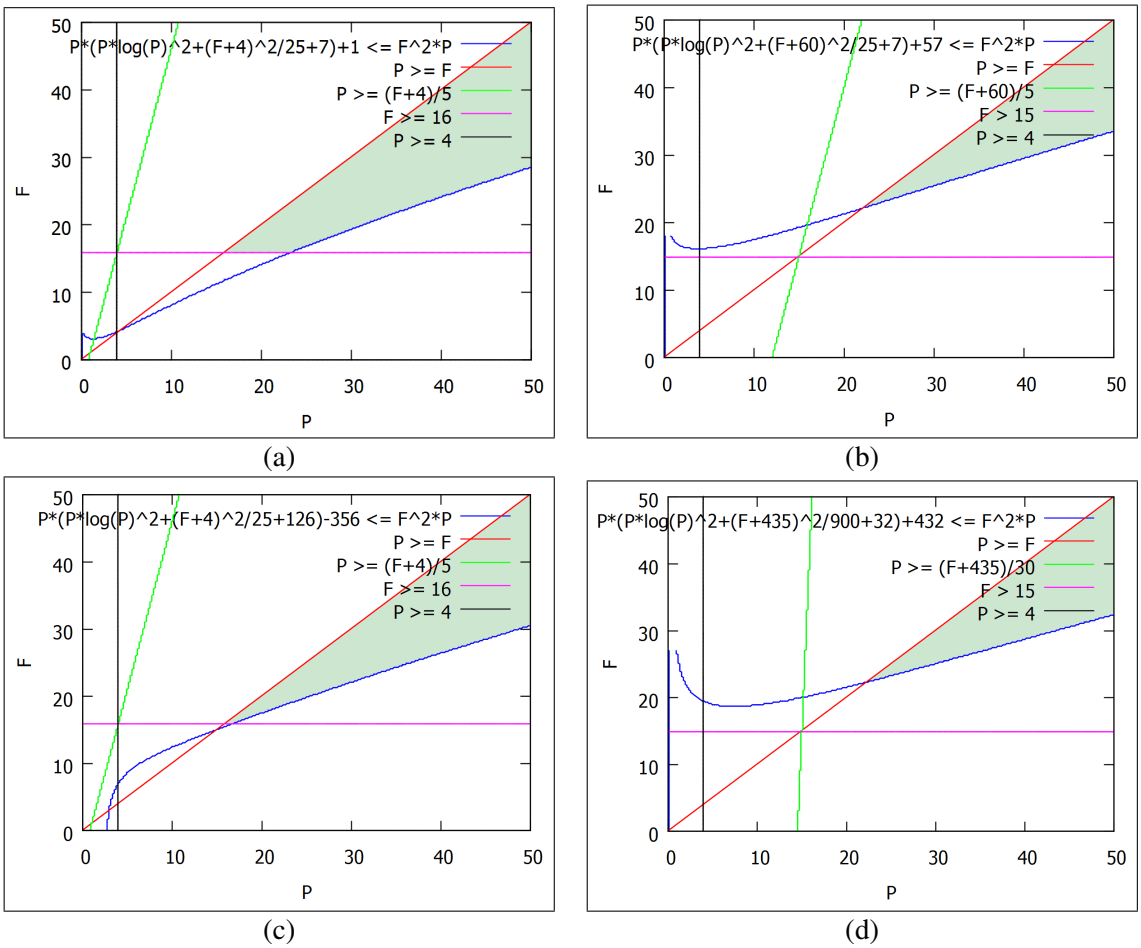


Figure 3.17: Analysis of the complexity of the method when $P \geq F \geq f$. (a) $k = 1$, $n = 5$ and $max\_iter = 1$. (b) $k = 15$, $n = 5$ and $max\_iter = 1$. (c) $k = 1$, $n = 5$ and $max\_iter = 120$. (d) $k = 15$, $n = 30$ and $max\_iter = 1$.

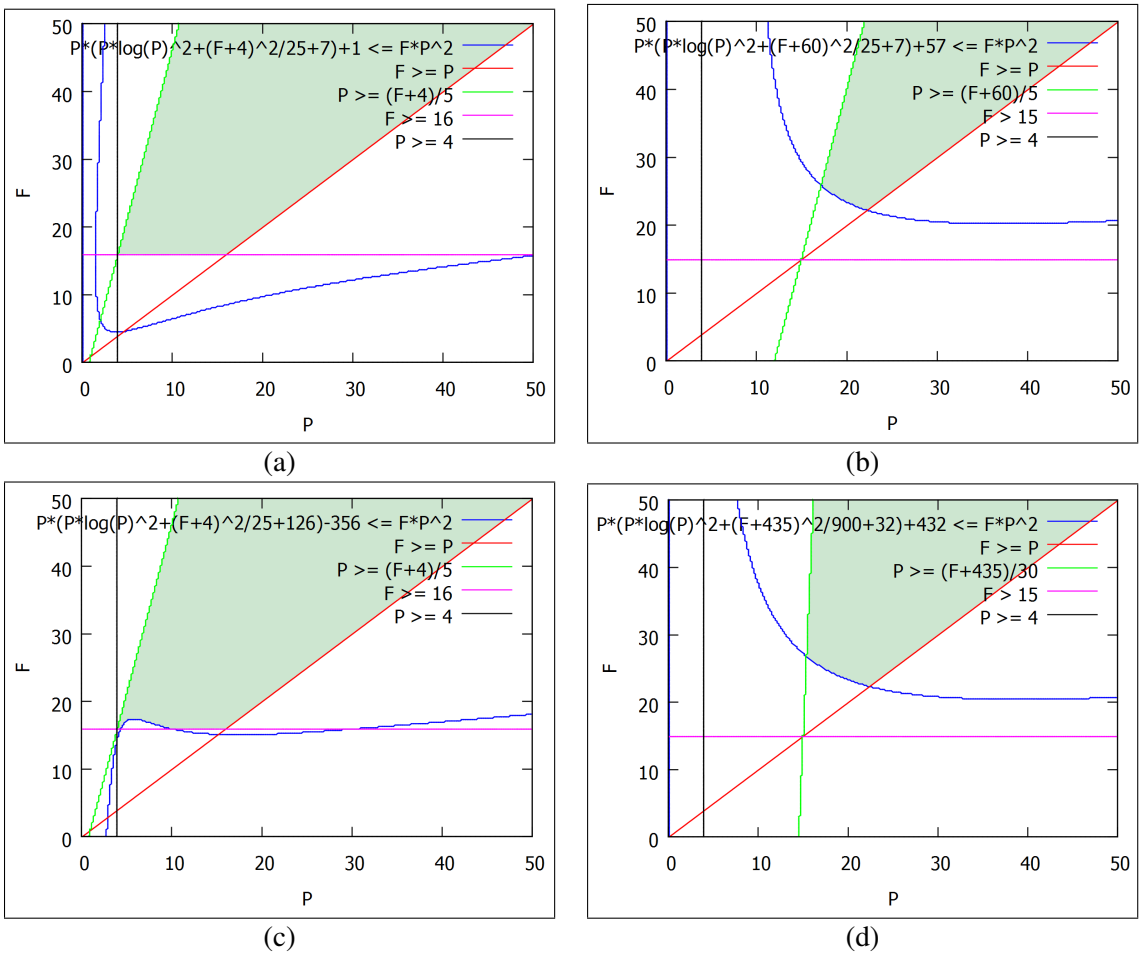


Figure 3.18: Analysis of the complexity of the method when $F \geq P \geq f$. (a) $k = 1$, $n = 5$ and $max\_iter = 1$. (b) $k = 15$, $n = 5$ and $max\_iter = 1$. (c) $k = 1$, $n = 5$ and $max\_iter = 120$. (d) $k = 15$, $n = 30$ and $max\_iter = 1$.

$max\_iter = 1$, RPFM performs better than the classical method for all valid points. The complexity (blue line) is not responsible for reducing the green area. Nonetheless, by increasing the number of overlapping frames, a region where the classical method performs better appears. In this situation, the system is also more sensible to the number of overlapping frames than the number of iterations. By setting max_iter to 120, RPFM still performs better than the classical method in almost every valid point. Finally, the effect of increasing the number of videos simply increases the valid region, as can be seen by the green line being almost vertical. This happens since f is smaller, for the same F.

Finally, the case where $F \geq f \geq P$ was also evaluated. The results are presented on Figure 3.19. When $k = 1$, $n = 5$ and $max\_iter = 1$, RPFM performs better for all valid points. The valid region happens to be small, but if one compares Figure 3.18(a) with Figure 3.19(a), it is possible to conclude that one complements the other. The behaviour that was observed on the previous cases are also observed in this one. The system is more sensible to the addition of overlapping frames than to the increase of the number of iterations.

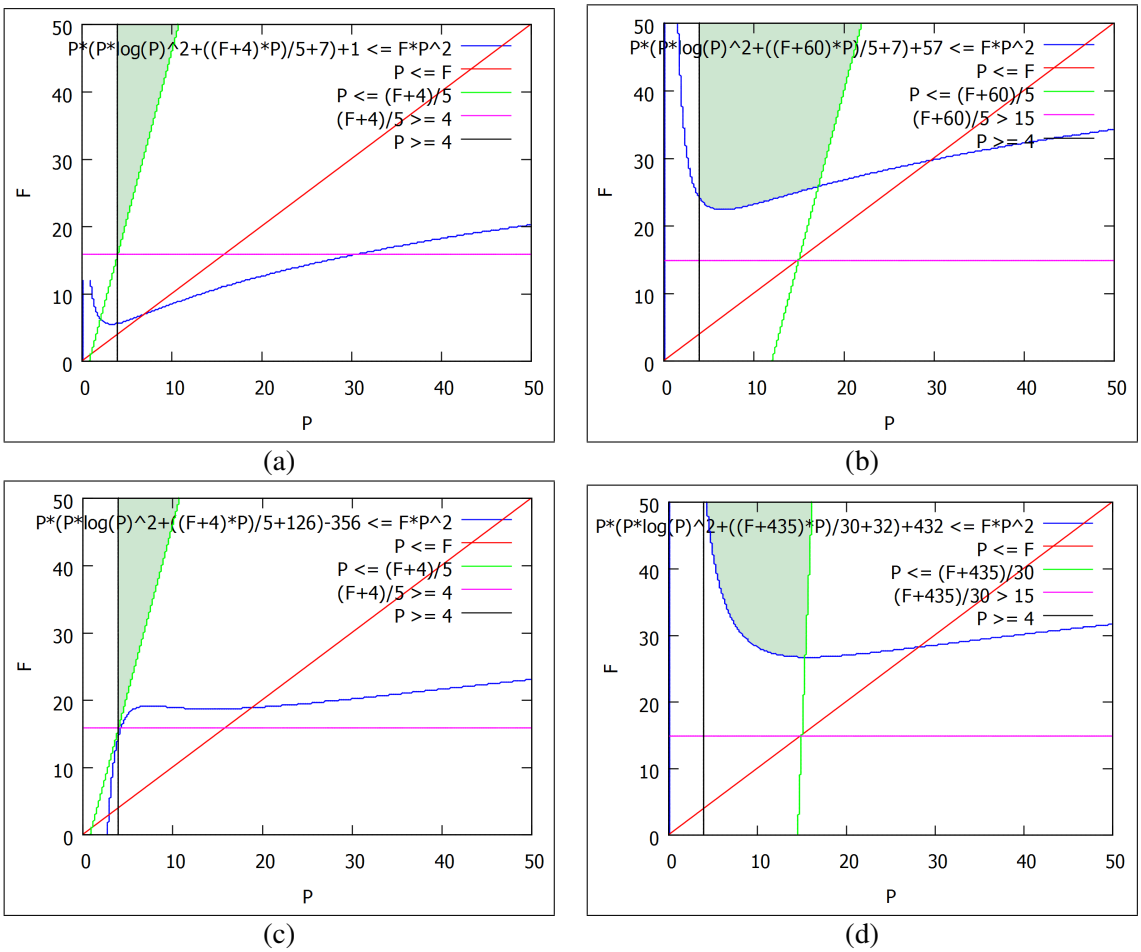


Figure 3.19: Analysis of the complexity of the method when $F \geq f \geq P$. (a) $k = 1$, $n = 5$ and $max\_iter = 1$. (b) $k = 15$, $n = 5$ and $max\_iter = 1$. (c) $k = 1$, $n = 5$ and $max\_iter = 120$. (d) $k = 15$, $n = 30$ and $max\_iter = 1$.

However, if one combines all the three cases into one, the region where RPFM performs better than the classical method is considerable. Again, this complexities are only realistic in the cases where it is possible to perform the method completely in parallel or, in other words, when the number of cores or nodes is greater or equal than n. Also, it is important to note that, if the method is to be implemented in a distributed system, the costs of communication should be taken into account.

Chapter 4

Distributed Architecture

Having a parallelizable SfM method is not enough, a scalable *web service* is required. The system must be able to adapt to a rapid increase in the number of users and, also, to longer videos. Finally, the system must be fault-tolerant. There are two approaches to scale a system:

1. *Vertical Scaling*: Also known as *scaling up*, consists of adding more resources to an existing node in the system, by adding CPUs or memory to a single machine.

2. *Horizontal Scaling*: Also known as *scaling out*, means adding more machines to a distributed system. It is possible to configure hundreds of small computers in a *cluster* to obtain aggregate computing power that can perform tasks that once would have required a supercomputer.

The approach chosen was the *horizontal scaling* since it is more fault-tolerant and makes it possible to share resources. If one machine fails before ending a computation, the task is rescheduled to another node. Also, as some nodes will be used for different requests, the system is more resource efficient.

When there is a need for more computational power, a new machine is added to the system accordingly. It is important to note that a physical machine can accommodate several nodes: a machine can host several virtual machines that, in turn, represent a node. With this idea in mind, it is possible to combine the two scale approaches since it might be desirable to improve the resources of the machines that are more compute-intensive and can become bottlenecks, therefore, improving response times.

In this Chapter, a description of a possible architecture to distribute the computation using the method of Chapter 3 is presented. Section 4.1 presents the technologies used for the implementation of the distributed system, Section 4.2 presents the architecture of the system, System 4.3 explains how to scale the system for more users and Section 4.4 states how to evaluate the system.

4.1 Communication Technologies

In order to distribute the computation, nodes need to communicate. Some need to schedule a task to be performed by another node, others need to signal they have concluded a task and others need to share information.

RabbitMQ is a lightweight message broker that is able to transmit messages between nodes in a fault-tolerant manner. In order to better understand RabbitMQ, it is best to start by defining some terms. A *producer* is a program that sends a message; a *queue* stores and sends messages, being essentially an infinite buffer; and a *consumer* is a program that waits to receive messages. These entities do not need to reside on the same machine.

A *producer* sends messages, limited to 255 bytes, to one or more *queues* and can continue doing other work. The *queue*, which is a FIFO buffer, eventually dispatches a message to the next *consumer* in sequence (round-robin) that subscribed to that *queue*. Round-robin is used since the nodes that subscribe to the same *queue* are supposed to run the same code and, therefore, take approximately the same time to finish. RabbitMQ is able to work with more complex models, such as the Publish/Subscribe model, however they are not needed nor desirable for this particular system.

If one *consumer* fails before finishing the task, the *queue* re-sends the message to another *consumer*. That way, it is easy to make the system fault-tolerant.

The system also needs to transfer videos between nodes. Since the videos can contain a considerable amount of data, it is not recommended for them to be transferred directly over TCP. These videos have an interesting property that can be exploited: they have overlapping frames among them.

Having that in mind, the BitTorrent protocol was used. BitTorrent is a peer-to-peer file-sharing protocol that is inherently distributed and fault-tolerant. Again, it is advisable to start by defining some terms related to the protocol. A *peer* is one instance of a BitTorrent client that wants to download a file but has only parts of it; a *seed* is a machine that possesses a part of the data and uploads it for other peers; and a *tracker* is a server that keeps track of which peers and seeds are in the swarm.

A *peer* that wants to share a file or group of files starts by creating a "torrent" file, which contains *metadata* about the tracker and the data to be shared. *Peers* that want to download a file must first obtain the torrent file, so they can connect to the tracker which, in turn, returns information about *peers* and *seeds* that possess the pieces of the file.

The main reason for choosing this protocol is that it can download files from its *peers*, not only from one computer, which reduces the load on the first *seeder*.

Although the BitTorrent protocol works well with large amounts of data, it is not ideal for small files. In order to transfer small files between nodes, such as torrent files and matrices, the TCP protocol is used. The reason to choose TCP over UDP is that TCP is reliable and guarantees the packets to be delivered in order.

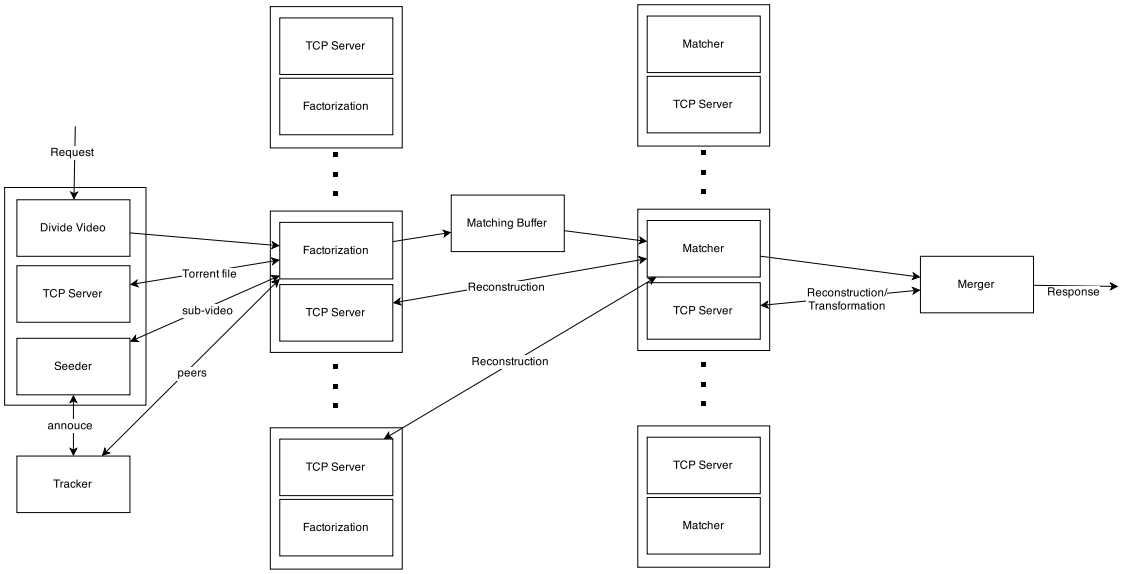


Figure 4.1: Architecture of the Distributed System

4.2 Architecture

The code was divided into seven modules:

1. Seeder
2. TCP Server
3. Divide Video
4. Factorization
5. Matching Buffer
6. Matcher
7. Merger

The whole system can be seen on Figure 4.1. As a simplification, the directed arrows between nodes represent RabbitMQ messages and the labelled bi-directional arrows represent TCP connections. The model abstracts the fact that a RabbitMQ message is sent to a *queue* before being delivered to the node. Nodes that are inside the same box are required to reside inside the same virtual machine.

The first two modules are auxiliary. The *Seeder*, as the name implies, is used to *seed* files using the BitTorrent protocol. The *TCP Server* is used to transfer small files between nodes.

When the distributed system receives a request for reconstructing a video, that request is handled by a *Divide Video* node that starts dividing the video as explained in Chapter 3. The video

must be decomposed on the form of frames and, as soon as a video $v^{(i)}$ is created, a torrent file with the *metadata* of the corresponding frames is also created, and its URL is sent to a *queue* q_0 to which the *Factorization* nodes subscribed. As soon as the torrent file is created, an HTTP request is sent to the tracker announcing what frames it is seeding.

When a *Factorization* node receives a message, it starts by downloading the torrent file. To do that, it reads the URL that was sent in the message and starts a TCP connection with a *TCP server*, residing in the same machine as the *Divide Video*, asking for the file. The *TCP server* receives a request for a file, divides the file in chunks and sends them to the *Factorization* node. Then, the *Factorization* node connects to the tracker and starts downloading the video.

When the first frame is completely downloaded, SURF features are extracted and, as soon as the next frame is available, the points are tracked. After the tracking process, the factorization method is applied and the matrices $W^{(i)}$, $S^{(i)}$, $M^{(i)}$ and $t^{(i)}$ are saved in one file. A message is then sent to *queue* q_1, to which the *Matching Buffer* node subscribed, with the URL of the file.

The *Matching Buffer* pairs the factorizations. As soon as two consecutive factorizations have finished, it starts a *Matcher* task. In order to do that, a message is sent to *queue* q_2 with the URLs of the files containing the matrices of two consecutive factorizations. A *Matcher* node, listening to *queue* q_2, is responsible for downloading the two files and then obtaining point matches between the two factorizations. The transformation between the two reconstructions is computed, using RANSAC, and a message is sent to *queue* q_3 with the URLs of the files with the matrices of the two factorizations and to the computed transformation and matches.

Finally, the *Merger* node is listening to the *queue* q_3 and, as soon as a message arrives, downloads the matrices from the two factorizations, the transformation between the two and the matches. All videos $v^{(i)}$, except for the first and the last one ($i \in [2, n-1]$), will arrive to the *Merger* twice. That happens because a video is sent to the *Matcher* twice, to be aligned with the previous video and with the next. As such, the *Merger* checks if it already saved the factorization before downloading.

As soon as the *Merger* receives all the transformations and factorizations it starts to reduce the noise. Then, all the factorizations are merged, and the result is sent to the client.

4.3 Scale

The first requirement was that the system should be capable of scaling with the number of frames. That requirement is partially achieved by adding more *Factorization* and *Matcher* nodes. The biggest problems are the *Divide Video* and specially the *Merger* node, which have a linear complexity with respect to the length of the video. In that sense, the requirement is not fully met and further optimizations should be pursued.

In order to scale with the number of users, the system of Figure 4.1 should be horizontally replicated. This means that more nodes of each type should be added proportionally to the number of users. Then, a *load balancer* is added to distribute the requests by the different *Divide Video* nodes.

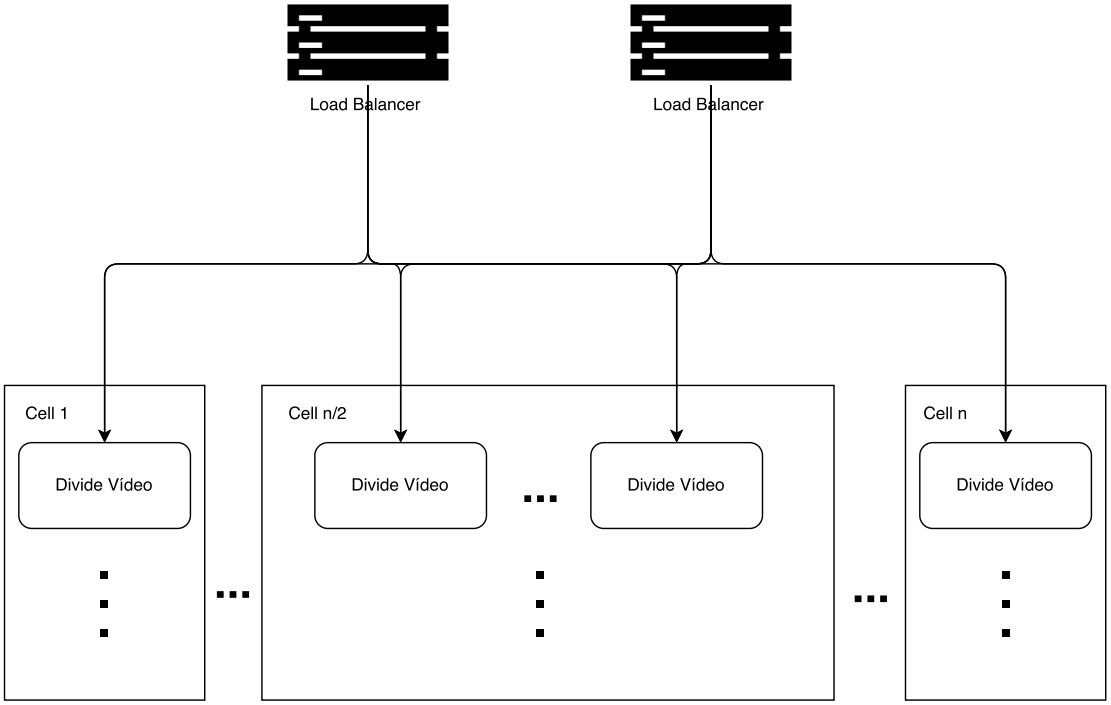


Figure 4.2: Cells and *load balancers* to scale with the number of users

A *load balancer* is a device that acts as a reverse proxy. In practice, it distributes the requests across different servers. It does that by listening on a port where external clients connect and forwards the request to one of the available servers, in this case, to a *Divide Video* node. It is important that the *load balancers* do not become a single point of failure. In order to achieve that, they are implemented in high-availability pairs and, if one fails, the other continues to do the work.

The system can be divided in different "*cells*". Let us define a *cell*, in the context of this thesis, as a cluster of nodes that implement the system of Figure 4.1. Each *cell* must be able to work independently, meaning that it must have its own RabbitMQ server and *tracker*. It is desirable that the communication between all entities inside each *cell* to be as fast as possible and, as such, they should reside inside the same local network. Since *cells* can be physically located in different places, the service can be improved for users with different geographic locations. A model of the *web service* can be seen on Figure 4.2.

4.4 Evaluation

The system from Figure 4.1 was implemented and a sample video was used to validate the system. As expected, it achieved the same results as the method described in Chapter 3. A more thorough evaluation was not possible due to resource limitations.

In order to properly evaluate the method, a vast number of computers is required. It would be interesting to evaluate the behaviour of the system by varying the load (number of users), the

length of the video and the number of nodes in the system, so an optimal ratio between the three variables would be found. This experiment would also prove the assumption of the system scaling with the number of users and length of the video.

Chapter 5

Conclusions and Future Work

In this thesis, a distributable method for performing Structure from Motion was presented and implemented. With the increase in the computational power and developments in distributed systems and cloud computing, it is natural that more complex methods will be available on the web. The increasing interest in distributed computer vision algorithms is an indication of this trend.

Machines still struggle in making sense of images. Still, the problem of reconstructing a rigid scene through images is almost solved. The next step is to reconstruct effectively a dynamic scene. Humans can easily create a mental model of the dynamic scene they are seeing, despite being hard to accurately state the measures of an object, and yet we can not really describe the process of how we do it. That is one of the reasons why the problem is yet to be solved.

However, it is still amazing how much information it is possible to extract from images that are nothing more than matrices of 0 to 255 integers. One step further would be to use context and previous knowledge in order to aid the reconstruction. By looking at only one image of a known object, we can easily extrapolate its full shape. Even more complex Computer Vision systems can still be fooled by light and reflections. To the knowledge of the author, it is still impossible to perform the reconstruction of reflective or transparent objects, and yet humans do that daily without effort. Current computers and algorithms are still far from being able to achieve such tasks, but are getting closer.

5.1 Discussion

The proposed method was able to perform reconstructions of real world objects, in challenging scenarios, even with the assumption of an orthographic projection. There are, still, a lot of limitations to the proposed system that are going to be covered in this section. Some limitations are common to image based methods and others to the method itself.

The greatest limitations of the method are derived by the assumption of orthographic projection, as it is not a good approximation of the perspective projection which is the one regular

cameras use. The orthographic projection does not account for the change in the size of an object as the camera moves towards or away from it. As it was noted on Subsection 3.3.6, some components of the object might be reconstructed with different scales. Also, the variation of the focal distance is not accounted for, meaning that the user can not zoom in or out while performing the video.

The SVD can deal with Gaussian noise in the measures. The method can, up to some extent, perform the reconstruction with noisy measures. Nonetheless, the tracking of the features is still a crucial stage of the whole process. The tracking method was never the focus of this thesis, there is a lot of work developed in that area, but the limitations must be accounted as they directly affect the results of the system. Highly reflective, transparent and textureless objects will not have satisfactory results, since the tracker will not be able to track effectively features at the surface of the object.

A small number of points, as accurate as they might have been reconstructed, do not convey enough visual information about the object. The method presented in this thesis does not provide a dense point cloud. However, the number of points is usually enough to recognise visually the object that was reconstructed. Also, at least four matches are required between consecutive videos in order to align the two.

It is still not possible to perform the reconstruction of the scene in real world measures. The reconstructions are, however, performed up to an arbitrary scale.

By using several reconstructions of the scene, in different poses, the errors of the reconstructions are more contained. This means that some reconstructions will be better than others. It was possible to use this information to improve the bad reconstructions and, therefore, the final results. It was shown that this approach is more robust to noise than the classical approach.

Also, this parallel approach allows the construction of a distributed system that is able to horizontally scale with the number of users and with the length of the video. This approach is more fault tolerant and suitable to implement as a *web service* than the classical method.

As the method is capable of tracking features instead of matching them, it overcomes the correspondence problem. Objects with a repeatable texture might prove hard, or even impossible to match points between two images. The rubik's cube was one such example, where the 123D Catch was not able to obtain satisfactory results. The tracking approach performs better in such cases.

Nonetheless, the system proved to be more usable than existing approaches, as it is easier and faster to film an object than to take dozens of photographs, or use a sensor that is wired to a computer.

5.2 Future Work

The system that was presented in this document is still a work in progress as there are a lot of improvements that need to be addressed. These improvements are crucial in order for the system to be viable for commercial or scientific usage.

Probably the most important improvement would be the implementation of the Han and Kanade's [HK02] perspective factorization method. With this method, the user would be able to zoom the camera during the video and the proportions of the resulting point cloud would be more realistic. The ability to zoom the camera might not be crucial, but the quality of the resulting point cloud is the most important aspect of this type of applications.

Another important aspect is the texture on the final model. The texture of the scene should be extracted and superimposed on the final model in order to be more attractive to the end user.

Following the topic of improving the quality of the final model, the resulting point cloud needs to be more dense. One way to do that is by performing stereo rectification and matching to extract more points. The motion matrices M are not enough for this process, since it relies on the epipolar constraint. The projection matrices are required and so the perspective factorization is required to be implemented. This process is well documented on the work of Pollefeys et al. [PSC+04].

The fundamental matrix is computed for each pair of frames by using the corresponding projection matrices and then, the frames are warped so that the epipolar lines coincide with the image scan lines. As the two frames are rectified, the search of matches becomes one dimensional and can be solved as a path search problem. Pollefeys et al. stated that "the goal of a dense stereo algorithm is to compute corresponding pixel for every pixel of an image pair". Having a denser point cloud, the final model is better reconstructed and with a lot more detail.

The developed system is still not fully automatic. The number of frames per $v^{(i)}$ is still being tuned manually for each reconstruction. The best approach to overcome this problem would be to use the accelerometer and gyroscope of the *smartphone* to divide the video based on the motion of the camera. This would limit the application to *smartphone* users. A more general approach was developed by Li et al. [LGD10], that optimizes the division of the video based on the reconstructions.

In the context of this application, it is expected that the user walks in a closed circle around the object he wants to reconstruct. This information should be used to improve the system's results. As the user closes the circle, the points might not be reconstructed exactly on the same place. This problem is known as Scale Drift and was presented on Subsection 2.1.3. It is possible to look into the motion matrix M for frames that "close" the circle around the object, in order to obtain matches. With these matches, the method proposed by Strasdat et al. [SMD10] could be implemented.

In order to improve the results obtained from the *hallucination* step, a more robust matrix factorization algorithm should be implemented. As an example, the method developed by Meng and De la Torre [MdlT13] is able to deal with missing values and, as there is no need to choose a sub matrix to factorize, it uses more information and, therefore, achieves better results. Also, their matrix factorization method does not assume a distribution for the error, as the SVD assumes a Gaussian distribution. In realistic scenarios, the tracking process is not guaranteed to obtain measures with a Gaussian noise distribution. In fact, it is likely that the measures have a Mixture of Gaussian noise distribution that the method of Meng and De la Torre is able to model.

Finally, and as it was stated on Subsection 2.1.2, the Bundle Adjustment should be applied as the last step of the system. This process refines the structure and motion improving the quality of the final model.

References

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, volume 32, pages 415–421, 1998.

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19:127–153, 2001.

[AFDM08] a. Angeli, D. Filliat, S. Doncieux, and J.-a. Meyer. Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words. *IEEE Transactions on Robotics*, 24(5):1027–1037, October 2008.

[AFS+09] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Brian Curless, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building Rome in a Day. pages 105–112, 2009.

[AKB08] Motilal Agrawal, Kurt Konolige, and MR Blas. Censure: Center surround extremas for realtime feature detection and matching. *Computer Vision–ECCV 2008*, pages 102–115, 2008.

[AMF03] Sergei Azernikov, Alex Miropolsky, and Anath Fischer. Surface Reconstruction of Freeform Objects Based on Multiresolution Volumetric Method. *Journal of Computing and Information Science in Engineering*, 3(4):334, 2003.

[Bas09] Rafael Bastos. *FIRST–Fast Invariant toRotation and Scale Transform: Invariant Image Features for Augmented Reality and Computer Vision*. VDM Verlag, 2009.

[BBX95] C L Bajaj, F Bernardini, and G Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 95), ACM Press*, pages 109–118, 1995.

[BDw06] B Y T I M Bailey and Hugh Durrant-whyte. Simultaneous Localization and Mapping (SLAM):. (September), 2006.

[BF05] A. Buchanan and A. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2(3):316–322, 2005.

[BMR+99] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5:349–359, 1999.

[BN78] James R. Bunch and Christopher P. Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31(2):111–129, 1978.

[Boi84] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation, 1984.

[BSLS06] Andrew Barton-Sweeney, Dimitrios Lymberopoulos, and Andreas Savvides. Sensor localization and camera calibration in distributed camera sensor networks. In *2006 3rd International Conference on Broadband Communications, Networks and Systems, BROADNETS 2006*, 2006.

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, 2006.

[CF13] Jim; Chandler and John Fryer. AutoDesk 123D Catch : How accurate is it? *Geomatics world*, (February):28–30, 2013.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, 2010.

[COC+13] Pedro Carvalho, Telmo Oliveira, Lucian Ciobanu, Filipe Gaspar, Luís F. Teixeira, Rafael Bastos, Jaime S. Cardoso, Miguel S. Dias, and Luís Côrte-Real. Analysis of object description methods in a video object tracking environment. *Machine Vision and Applications*, 24:1149–1165, 2013.

[DCSN04] Antonio G. Dopico, Miguel V. Correia, Jorge a. Santos, and Luis M. Nunes. Distributed Computation of Optical Flow. *Computational Science - ICCS 2004*, 3037:380–387, 2004.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13, 2008.

[DM15] Salam Dhou and Yuichi Motai. Dynamic 3D surface reconstruction and motion modeling from a pan-tilt-zoom camera. *Computers in Industry*, 70:183–193, 2015.

[DR04] D. Devarajan and R. J. Radke. Distributed Metric Calibration of Large Camera Networks. In *In Proc 1st Workshop on Broadband Advanced Sensor Networks*, volume 2, pages 380–403, 2004.

[DRC06] Dhanya Devarajan, Richard J. Radke, and Haeyong Chung. Distributed metric calibration of ad hoc camera networks, 2006.

[ELF97] D.W. Eggert, a. Lorusso, and R.B. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290, 1997.

[EM94] H Edelsbrunner and E P Mucke. 3-dimensional alpha-shapes. *ACM Transactions on Graphics*, 13:43–72, 1994.

[FGPS06] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar. Distributed localization of networked cameras. *2006 5th International Conference on Information Processing in Sensor Networks*, 2006.

[FS12] By Friedrich Fraundorfer and Davide Scaramuzza. Visual Odometry. (June), 2012.

[GKS+10] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, Udo Frese, and Christoph Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 273–278, 2010.

[Gra01] Claus Gramkow. On averaging rotations. *Journal of Mathematical Imaging and Vision*, 15(1-2):7–16, 2001.

[Haj05] Levente Hajder. An iterative improvement of the Tomasi-Kanade factorization. *Third Hungarian Conference on Computer Graphics and Geometry, November*, pages 37–42, 2005.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points, 1992.

[HK02] Mei Han and Takeo Kanade. A perspective factorization method for euclidean reconstruction with uncalibrated cameras. *The Journal of Visualization and Computer Animation*, 13(4):211–223, 2002.

[HP87] C. G. Harris and J. M. Pike. 3D Positional Integration from Image Sequences. *Procedings of the Alvey Vision Conference 1987*, pages 32.1–32.4, 1987.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. 2006.

[KBWT14] Ryan Kennedy, Laura Balzano, Stephen J. Wright, and Camillo J. Taylor. Online algorithms for factorization-based structure from motion. *2014 IEEE Winter Conference on Applications of Computer Vision, WACV 2014*, (September):37–44, 2014.

[KSBW03] Timo Kohlberger, Christoph Schnörr, Andrés Bruhn, and Joachim Weickert. Domain Decomposition for Parallel Variational Optical Flow Computation. *DAGM-Symposium*, 2781:196–203, 2003.

[KSO04] R Kolluri, J R Shewchuk, and J F O'Brien. Spectral Surface Reconstruction from Noisy Point Clouds. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 71:11–21, 2004.

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1988.

[LA06] H Lee and H Aghajan. Collaborative node localization in surveillance networks using opportunistic target observations. In *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, page 18, 2006.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm, 1987.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. *2011 International Conference on Computer Vision*, pages 2548–2555, November 2011.

[LGD10] Ping Li, Rene Klein Gunnewiek, and Peter De With. Detecting critical configurations for dividing long image sequences for factorization-based 3-D scene reconstruction. *Lecture Notes in Computer Science (including subseries Lecture Notes in*

Artificial Intelligence and Lecture Notes in Bioinformatics), 5995 LNCS:381–394, 2010.

[LK81] BD Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *IJCAI*, 1981.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LWW12] Jingyu Li, Yulei Wang, and Yanjie Wang. Visual tracking and learning using speeded up robust features. *Pattern Recognition Letters*, 33(16):2094–2101, 2012.

[MdlT13] Deyu Meng and Fernando de la Torre. Robust Matrix Factorization with Unknown Noise. *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1337–1344, 2013.

[MGVP14] A. Masiero, A. Guarnieri, A. Vettore, and F. Pirotti. An ISVD-based Euclidian structure from motion for smartphones. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5(June):401–406, 2014.

[MK10] Shivaji P Mirashe and N V Kalyankar. Cloud Computing. *Communications of the ACM*, 51(7):9, 2010.

[MM98] R. Mencl and H. Muller. Graph-based surface reconstruction using structures in scattered point sets. *Proceedings. Computer Graphics International (Cat. No.98EX149)*, 1998.

[NW99] J Nocedal and S J Wright. *Numerical Optimization*, volume 43. 1999.

[PBK+12] Timo Pylvänäinen, Jérôme Berclaz, Thommen Korah, Varsha Hedau, Mridul Aanjaneya, and Radek Grzeszczuk. 3D city modeling from street-level data for augmented reality applications. *Proceedings - 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2012*, pages 238–245, 2012.

[PK94] Conrad J. Poelman and Takeo Kanade. A paraperspective factorization method for shape and motion recovery. In *Proceedings of the 3rd European Conference on Computer Vision, Stockholm, Sweden*, pages 97–108. Springer, 1994.

[PSC+04] Marc Pollefeys, Computer Science, North Carolina, Chapel Hill, L U C V A N Gool, Maarten Vergauwen, Frank Verbiest, Kurt Cornelis, J A N Tops, and Reinhard Koch. Visual Modeling with a Hand-Held Camera. 59(3):207–232, 2004.

[Rad10] Richard J. Radke. A Survey of Distributed Computer Vision Algorithms. In *Handbook of Ambient Intelligence and Smart Environments*, pages 35–55. 2010.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, pages 1–14, 2006.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. *2011 International Conference on Computer Vision*, pages 2564–2571, November 2011.

[SEE+12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[SMD10] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Scale Drift-Aware Large Scale Monocular SLAM. *Robotics: Science and . . .* , 2010.

[SSS+08] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3D architectural modeling from unordered photo collections. *ACM Transactions on Graphics*, 27(5):1, 2008.

[ST94] J Shi and C Tomasi. Good features to track. *. . . , 1994. Proceedings CVPR'94., 1994 IEEE . . .* , 1994.

[Sze10] Richard Szeliski. Computer Vision : Algorithms and Applications. *Computer*, 5:832, 2010.

[THZ13] Renoald Tang, Setan Halim, and Majid Zulkepli. Surface Reconstruction Algorithms: Review and Comparison, 2013.

[TK92] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, November 1992.

[TM08] Tinne Tuytelaars and Krystian Mikolajczyk. Local Invariant Feature Detectors : A Survey. 3(3):177–280, 2008.

[TMHF00] Bill Triggs, Philip F Mclauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle Adjustment — A Modern Synthesis. *Vision algorithms: theory and practice. S*, 34099:298–372, 2000.

[Tri96] Bill Triggs. Factorization methods for projective structure and motion. *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages845(June):845–851, 1996.

[TV11] Roberto Tron and Reńe Vidal. Distributed computer vision algorithms through distributed averaging. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 57–63, 2011.

[TYAB01] Lorenzo Torresani, Danny B. Yang, Eugene J. Alexander, and Christoph Bregler. Tracking and modeling non-rigid objects with rank constraints. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1, 2001.

[Ume91] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[WKYW13] Ying-mei Wei, Lai Kang, Bing Yang, and Ling-da Wu. Applications of structure from motion: a survey. *Journal of Zhejiang University SCIENCE C*, 14(7):486–494, July 2013.

REFERENCES

[XCK04] Jing Xiao, Jinxiang Chai, and Takeo Kanade. A Closed-Form Solution to Non-Rigid Shape and Motion Recovery. *In European Conference on Computer Vision*, pages 573–587, 2004.

[ZSXW15] Zhong Zhou, Feng Shi, Jiangjian Xiao, and Wei Wu. Non-Rigid Structure-From-Motion on Degenerate Deformations With Low-Rank Shape. *IEEE Transactions on Multimedia*, 17(2):171–185, 2015.