Università di Pisa

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Computer Engineering

Laurea Specialistica in Computer Engineering

3D Environments Reconstruction using 360 Videos

Supervisors:

Marco AVVENUTI

Francesco BANTERLE

Massimiliano CORSINI

Candidate:

Andrea BECONCINI

Academic Year 2016 17

Abstract

360°degrees or full spherical images are gaining a huge interest in different fields such as autonomous driving, cinematography, augmented reality (AR), and virtual reality (VR).

Computer vision research addressing spherical images is less popular than the one that considers traditional perspective cameras. This new kind of devices have some advantages with respect to standard cameras, for example, it allows users to capture an entire environment in a single shot.

In this work, we developed a structure from motion (SfM) pipeline for full spherical cameras composed of two main parts: camera poses estimation and dense point cloud reconstruction. This pipeline employs frames captured using a 360°video-camera in the equirectangular format.

Our contribution includes: a visual-based frame filter that selects frames to be used for motion estimation, a novel SfM pipeline implementation in MATLAB, and an adaptive window matching procedure for point cloud densification.

We tested the performance of our work both with synthetic 3D scenes and with real sequences captured with a Ricoh Theta S camera.

Contents

Chapter 1

Introduction

The growing interest of the general public in spherical cameras technologies, which were research-only exclusives once, has made them less expensive and more available. For example, omnidirectional cameras by GoPro have been extremely popular amongst sportsmen who publish their activities on social networks.

These devices have been employed in robotics since their first appearance to help navigate the robot in a real-world environment, and nowadays for autonomous driving. Furthermore, they can be exploited to deliver a virtual or augmented experience for augmented and virtual reality (AR/VR).

In this work, we designed a *structure from motion* (SfM) pipeline for full spherical cameras. SfM is a well-known topic in computer vision; it addresses the problem of recovering the structure of 3D environments from a sequence of images taken from different viewpoints.

SfM research (and computer vision in general) has targeted perspective cameras because these type of devices have always been more common. However, the increase of interest in these cameras has started a shift of interest towards employing them for research.

1.1 Omnidirection Cameras

Large field of view (FOV) photography employs fisheye lenses that are capable to cover wider scene compared to traditional cameras' hardware. The applications

(a) Fisheye.

(b) Ominidirectional (catadioptric).

Figure 1.1: Some examples of fisheye pictures.

for this types of equipment range from architectural or landscape photography to academic use for studies related to astronomic, meteorology, computer graphics, etc. Photographers can also choose fisheye lenses for the characteristic distortion these devices introduce and that provides more importance to foreground objects (see Figure 1.1 (a) for an example of a picture taken with fisheye lenses equipped camera). An example of wide FOV photography equipment is the GoPro's camera series. GoPro Inc. produces action cameras, small digital devices for recording videos and taking pictures in harsh environments; their products have gained quite some success among the consumer market, also thanks to their marketing campaign that successfully associated their product to extreme sports events. Action cameras like GoPro's exploit wide FOV lenses that help to capture larger scenes and make the devices fit to be worn.

Apart from wide FOV hardware, there is another class of cameras that take panoramic photography to a new level: the 360°full spherical devices, capable of capturing images in all directions with no blind corner. This class of devices employs several image sensors and integrated stitching software to automatically produce panoramic pictures. This is the case for the Ricoh Theta S [44] we used in this work (Figure 1.2).

Together with consumer hardware, some off-the-shelf solutions for the profes-

Figure 1.2: The Ricoh Theta S 360°camera.

sional market have started to appear. They include the Insta360 Pro that can capture full spherical videos or pictures at 8K resolution. Another example is the GoPro Omni. This camera is a simple rig that contains 6 traditional GoPro devices that comes with specialized software to enables the 360°videos creation. Note that the interest in full spherical media creation is driven by the introduction of new immersive *head-mounted display* (HMD) such as the Oculus Rift, the HTC Vive, the Sony PlayStation VR, etc.

1.2 Benefits of Full Spherical Cameras

Because of the increased *field of view* (FoV), panoramic cameras can capture a larger amount of data compared to traditional devices. For example, the two fisheye lenses in the Ricoh Theta permit to use both the front and back image information and this may improve the quality of motion estimation.

Another advantage of full spherical cameras is that they do not require a calibration phase for intrinsic parameters estimation. Camera calibration is an essential requirement for most computer vision applications, which estimates several parameters such as focal length, image sensor size, pixel density, and lens distortion model. When dealing with full spherical cameras, we can assume the image is taken from a unitary sphere. Therefore, the knowledge of the internal camera parameters is not required. Further details about the camera model and the parameters can be found in Chapter 2.

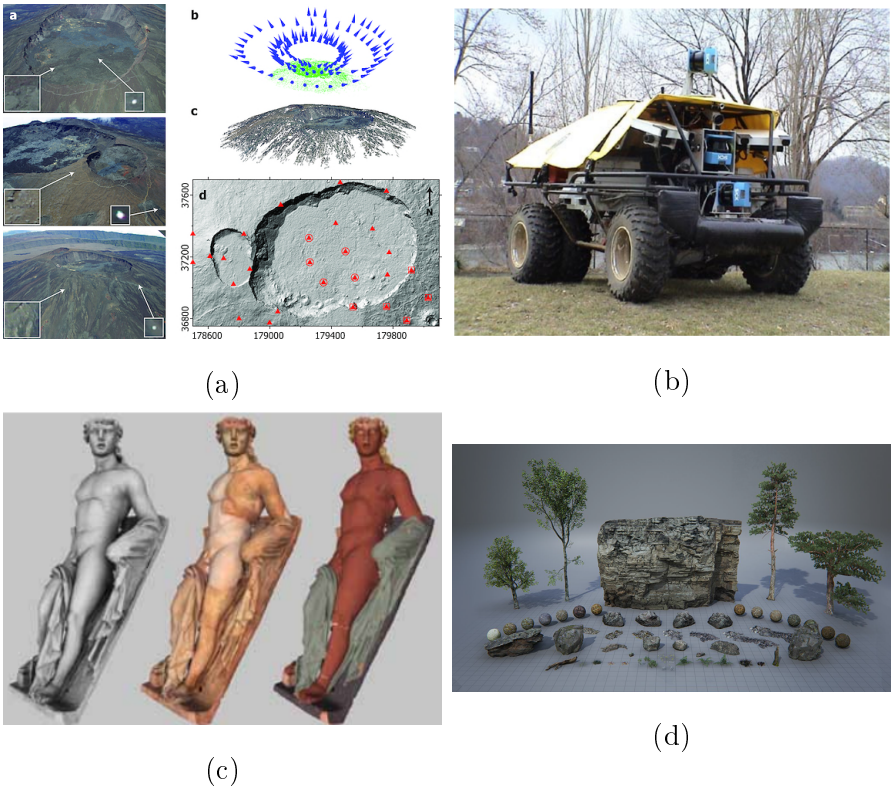
1.3 SfM Applications

SfM is a well-known topic in the computer vision community; it started after the landmark papers by Longuet-Higgins [25] and Nister et al. [32]. The problem can be described as the reverse of image formation [47], as it targets the reconstruction of the environment and camera poses given a set of two or more images. Some of the first application for this technology included robotic research, like navigation systems intended for rover explorations [32, 4]. In fact, NASA has supported many research because of the need for navigation systems not affected by wheels slippage on uneven terrains. SfM is used in geographical data acquisition too [6, 48, 17] as an alternative to other methods that employ specialized hardware and are generally more expensive. Furthermore, there are applications for cultural heritage conservation because environment reconstructions can help in case of restoration of historical finds [20]. Finally, the game industry is yet another field that can benefit from SfM: real environments can be reconstructed with SfM first and then improved by artists. Some game engine, as the popular Unreal Engine, includes photogrammetry software that exploits SfM techniques.

1.4 Our Contribution

In this work, we propose a novel SfM pipeline for full spherical cameras that estimates the camera poses and produces a dense point cloud representing the environment's geometry. We address full spherical videos with equirectangular frame format. To reach our goal we developed the following components:

- a frame selector that extracts the spherical video's frames to be used for pose estimation;
- an adaptive block-matching algorithm for disparity map creation optimized for equirectangular images;
- the complete pipeline implementation in MATLAB that includes all the adjustment needed for traditional SfM routines to work with full spherical images.



(a) (b)

(c) (d)

Figure 1.3: Some examples of SfM applications: geoscience can benefit of SfM, and computer vision in general, as an alternative to specialized and more expensive measuring tools [17] (a). Real-time applications of SfM are employed as part of the control software in autonomous driving vehicles [34] (b). SfM and dense stereo matching are used for cultural heritage conservation [40] (c). with the aim of photogrammetry, game developers can create realistic environments; SfM is one of the cornerstones of photogrammetry (d)[1].

[1]Image from `www.unrealengine.com/en-US/blog/imperfection-for-perfection`

Chapter 2

State of the Art

In this chapter, we define formally the VO problem, we describe a generic perspective SfM pipeline, and then we introduce the main challenges and issues when employing full spherical cameras for VO. During this description, we refer, time by time, to the relative state of the art algorithms/methods.

2.1 SfM, VO, and SLAM

SfM is a long-studied topic in computer vision. Starting from an input of photographs taken by one or multiple cameras, SfM's main goal is to compute the cameras' poses and to reconstruct the 3D environment's geometry captured by those cameras [43]. Some of the first works are the paper by Longuet-Higgins [25], whose equations are fundamental in epipolar geometry, and the paper by Tomasi et al. [45], who used orthographic photographs to estimate the shape of a 3D object. SfM is a general term, and it also includes *visual odometry* (VO). This is the problem of recovering the motion of an agent equipped with a camera rig in a 3D environment. Typically, VO deals with an ordered set of images such as video sequences, and it uses them to compute the ego-motion in real-time. Nister et al.[34] introduced the term *visual odometry* for the first time.

Another field of robotic/computer vision research that is very close to VO is *simultaneous localization and mapping* (SLAM). SLAM targets the problem of creating a map of the environment where a camera-equipped agent navigates and

simultaneously estimating its path. Scaramuzza[36] pointed out that while VO is more focused on local motion estimation, SLAM's goal is to obtain a global consistent estimation of the agent movements. In order to reduce errors, SLAM keeps track of a visited path and can decide when the agent has come back to a previously visited location. This extra step in SLAM's pipeline is called *loop closure* and provides an additional constraint used to reduce errors in both the agent's path and environment reconstruction. Durrant-Whyte et al.[4] firstly introduced the term SLAM.

2.2 Densification: from Sparse to Dense Point Cloud

A sparse point cloud is the product of most SfM pipelines; some keypoints are tracked in each photograph and are then used by a motion estimation algorithm. These keypoints are triangulated and their corresponding world points are available during ego-motion estimation. A set of triangulated keypoints is a sparse point cloud around the camera's path. This set provides a first approximation of the environment that we want to reconstruct. Typically, starting from this information, it is possible to obtain a complete reconstruction of a real-world environment by applying a so-called *Multi-View Stereo* algorithm. The goal of Multi-View Stereo[41] is to reconstruct a complete 3D object model from a collection of images taken from known camera poses. Many of these algorithms obtain a dense point cloud starting from an initial sparse reconstruction, and then estimate the final surface according to this dense point set.

Our approach consists in densifying the point cloud by computing, for each pair of consecutive images, a depth map. These depth maps are merged to obtain a dense set of reconstructed points. At this point, the surface can be reconstructed by applying a surface fitting algorithm such as the standard Poisson reconstruction [19]. A depth map is an image in which each pixel stores a depth value; i.e., the distance from the optical centre of the camera to an object in the scene. The depth map estimation involves the calculation of a *disparity map* from an images pair. This is an image of the distance in pixels from a point in a reference image

and its corresponding point in the other one. Higher is the disparity of a given pixel, higher is the parallax and hence the depth of that pixel. The computation of a disparity map is usually based on a similarity metric such as *sum of absolute distances* (SAD), *sum of squared distances* (SSD), *normalized cross correlation* (NCC), etc. These provide a way to compute the corresponding point in the second image, thus enabling the computation of the disparity for every point in the reference image. Figure 2.1 shows an example of a disparity map.

Figure 2.1: An example of a disparity map from the Middlebury dataset [13]: On the left, the input photograph. On the right, the corresponding depth map of the photograph on the left.

2.3 Omnidirectional and Full Spherical Cameras

Omnidirectional cameras are characterized by a wide field of view. Indeed, many of these cameras can take pictures with a 180°view angle or even wider.

There are several ways to obtain panoramic images, they include:

- perspective cameras and image stitching;
- catadioptric cameras;
- dioptric cameras;
- hybrid approaches.

Perspective cameras can take panoramic pictures with the aid of software stitching. As the first step, we take several photographs with many cameras or by simply moving the same camera in order to cover most of a scene. Then,

a stitching software merges all images into a single one. To use a traditional camera and a stitching software is the cheapest way to obtain panoramic images because it does not need any kind of specialized hardware. However, this process is cumbersome (a lot of manual work is required). Note that some smartphones provide built-in camera 360°acquisition modes, but the final quality may be not satisfactory due to alignment artefacts. An example of this approach is the Hugin Panorama Photo Stitcher [14], an open source software for merging several perspective images in a single panoramic picture. The SpheroCam by SPHERON-VR AG [42] is an alternative to the latter approach to panoramic images; it uses a specialized hardware composed of wide angle lenses equipped camera mounted on a rotating support, which rotates around the optical center of the camera. Therefore, this camera can capture high quality full spherical and it addresses the professional fields of geographical analysis, forensic investigation, cultural heritage, etc.

A catadioptric camera can be obtained by coupling a perspective camera and a mirror, which is mounted in front of the camera. Such setup takes a picture of the mirror that reflects the surrounding. The mirror of a catadioptric system may have several shapes. The most common one is the hyperbolic profile that creates a single centre of projection for every ray coming to the mirror.

A traditional camera can also be adapted to work as a panoramic one by adding a fisheye lens capable of refracting lights from wide angles towards the image sensor. These setups are called dioptric cameras.

Apart of perspective cameras coupled with stitching software, none of the previous approaches can take full spherical panoramic photographs. Moreover, full spherical videos cannot be shot with perspective cameras either.

The last approach, a hybrid one, exploits several sensors using fisheye lenses and software stitching to capture full spherical panoramic images in a single shot. This enables full spherical video capturing as well. The camera (that is composed of several image sensors) take multiple overlapping pictures simultaneously. Then, an on-camera stitching software composes the data in a single image.

In all our experiments we used the Ricoh Theta S camera that is a hybrid full

spherical camera composed of two fisheye lenses with a FOV greater than 180°; see Figure 1.2.

2.4 VO Problem

As we described in the previous chapter, the VO's goal is to recover the camera trajectory while it is moving in the environment. Following Scaramuzza and Fraundorfer[36], we introduce the notations that we are going to use for the rest of this work. Let first assume time is sampled in a sequence of time instants k; $I_{0:(n-1)}$ is the set of n input frames, where I_k is a picture taken by the camera at time k. We define $C_{0:(n-1)}$ as the set of camera positions such that C_k is the position at the k-th time. If we call T_k the rigid body transformation of the camera between two consecutive time instants (i.e., $k-1$ and k), then we have the following relation:

$$C_k = C_{k-1}T_k \tag{2.1}$$

where T_k is defined as

$$T_k = \begin{bmatrix} R_k & \mathbf{t}_k \\ 0 & 1 \end{bmatrix},$$

where R_k is a 3-by-3 rotation matrix (i.e., the rotation of the camera at time k), $\mathbf{t}_k$ is a column vector (i.e., translation of the camera at time k). Therefore, the goal of VO is to estimate both R_k and $\mathbf{t}_k$ for each time k and to compute each C_k accordingly to equation 2.1. Note that the initial position C_0 can be set arbitrarily.

Equation 2.1 is the core of VO, that allows us to compute the camera local movement. Therefore, we can estimate its location in every instant of time. However, the equation above contains also the most insidious practical challenge of VO; i.e., the error accumulation phenomenon known as *drift*. Every new position C_k can introduce an error factor that affects the next computations. The result is a global error that grows as the number of estimations increases. Since early works such as Harris et al. [10] and Moravec [32], the VO research community has focused on error minimization techniques to reduce drift. This tackles the

precise estimation problem either by employing more accurate local motion estimation and by introducing an optimization step to refine camera locations. The set of techniques for reducing the drift (after the camera poses are computed) is typically called *bundle adjustment* [46].

2.5 Perspective SfM

The SfM literature is vast, but most of the approaches present pipelines similar to the one in Fig. 2.2. The main steps are: to compute the relative motion for each image pair, to compose these motions for obtaining the absolute camera position and orientation, and to run a drift reduction procedure. The differences are in the type of input data, motion estimation algorithm, optimization procedure and additional constraints considered. For generic SfM researches, the input data is a set of traditional pictures of the same environment from different points of view. The pictures can be taken by different cameras with unknown parameters and in different time instants.

Visual Odometry is based on SfM techniques, in this case, the input data is usually a sequence of images from a video stream. The image capturing devices can vary: Moravec used a sliding camera that could capture stereo images [32] while Matthies et al. equipped their robot with an actual stereo rig [30]; in [34] Nister et al used a single perspective camera. The computer vision literature refers to the single-camera studies case with the term *monocular VO* while it uses *stereo VO* to describe the work done with stereo equipment.

Even though perspective cameras have been the first choice in many studies, the researchers used other devices, like the ones described in 2.3.

We use the terms *perspective* or *traditional SfM* to describe SfM pipelines designed for perspective cameras.

A great resource about the state of the art for VO and SfM is the work by Scaramuzza and Fraundorfer, [36, 37].

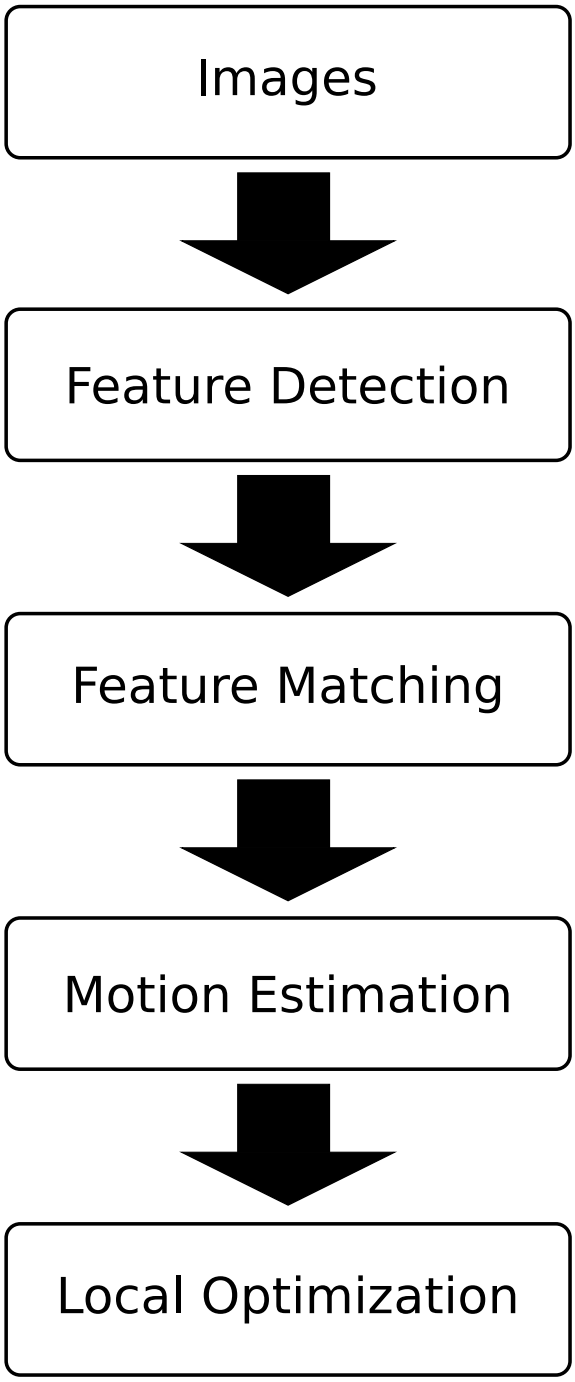


Figure 2.2: Perspective SfM pipeline block diagram.

2.5.1 Motion Estimation

We use the term local motion to indicate the camera movement between two consecutive images while, on the other hand, we define global motion as the overall camera's path. The local motion estimation step is fundamental in a SfM pipeline, its goal is to find the rigid body transformation composed of R_k and t_k as we described in Section 2.4. All the local motion estimation methods described in the literature so far are based on the correspondences found in two consecutive images but, depending on the specific camera rig (stereo imaging systems or monocular), the type of correspondences and their matching procedure, we may have several choices for the actual way to estimate local motion. There are two main families of point matching algorithms: *feature-based* and *appearance-based* (also known as *global-methods*). While the former ones utilize repeatable features matching between the images, the latter ones rely on pixel's intensity information; they are simpler but also slower. Most of the most recent VO pipelines use feature-based methods because of their speed and robustness. A VO implementation that employs intensity based techniques is [34], while [29] is an example of featureless motion estimation method.

Each of the feature points found by the motion estimation phase can be either a 3D world-point feature or a 2D image-point one, therefore, there are three possible combinations that provide just as many different matching and motion-estimation procedures:

- 2D-to-2D: both feature sets are composed of image points. The matching metric can be a simple euclidean distance between feature descriptors and the motion estimation can be solved by estimating the *essential matrix* (see section 2.5.1 for details);

- 3D-to-3D: both feature sets contain world-points features and motion estimation is performed by solving an alignment problem;

- 3D-to-2D: the previous image's feature set is composed of world points while the current one's feature set contains their projections. In this case,

the motion is estimated by solving a problem called Perspective-n-Points (PnP).

Since the second and third approaches deal with 3D points, they are usually best suited for stereo rigs (especially the 3D-to-3D case). Obviously, we can still use the 3D-to-2D approach in the monocular VO case by simply triangulating corresponding 2D image features in consecutive frames. In fact, in order to obtain the first 3D feature set, we can compute the relative motion between the first two images with the 2D-to-2D techniques and then obtain the rest of the camera's path, as we said, by solving the PnP problem.

Essential Matrix

The essential matrix E is defined as

$$\mathbf{p}'^{\top} E \mathbf{p} = 0, \tag{2.2}$$

where $\mathbf{p}$ and $\mathbf{p}'$ are, respectively, the normalized corresponding feature coordinates in image I_{k-1} and I_k. The normalized coordinates are defined as

$$\mathbf{p} = K^{-1}\mathbf{m}, \tag{2.3}$$

where K is the intrinsic parameter matrix, and $\mathbf{m}$ is an image point. The essential matrix contains the geometric information that describes the relative location of a camera respect to another one up to an unknown scale factor for the translation vector. In particular, we have:

$$E_k = \lambda \hat{t}_k R_k,$$

where λ is an unknown scale factor, and $\hat{t}_k$ is the skew-symmetric form for the cross product of vector $\mathbf{t}_k$. In order to extract the local motion given the two set of corresponding features in two consecutive images, we have to estimate the essential matrix and then extract the rigid body transformation out of it. For the E estimation part we can employ the Longuet-Higgins' 8-points algorithm [25]: the equation 2.2 provides a constraint we can exploit for computation, in fact, for each correspondence, we can rewrite the equation as

$$\begin{bmatrix} p_1 p'_1 & p'_1 p_2 & p'_1 & p_1 p'_2 & p_2 p'_2 & p'_2 & p_1 & p_2 & 1 \end{bmatrix} E = 0.$$

To find the nine unknowns, we need 8 non-coplanar feature matches that provide as many independent equations. In practice, we use more than 8 points, i.e., we obtain a overdetermined system that we solve in the least square sense.

Once we estimate E, there are four possible cases for the relative position of the two cameras and each world points. We are interested in the one that presents both cameras facing toward the triangulated world point; i.e., this point has to be in front of both cameras. We can determine the correct configuration among the four by testing each of them using triangulation.

Relative Scale

As we have already pointed out in the previous section, every monocular SfM pipeline that works with 2D-to-2D feature correspondences can estimate the local motion up to an unknown scale factor. There is no way to extract the translation magnitude from two sets of features (it is still possible to recover the reconstruction scale if we are given some world measure of the environment). Therefore, if we can not derive the local translation length with some other non-visual techniques (such as wheel odometry, GPS, accelerometer, etc.). We have to keep this unknown scale for the motion estimation and environment reconstruction.

On the other hand, if we have to compose the motion of more than two poses, as VO does, we need to set the first local translation magnitude arbitrarily and express all the other movements relative to this first one. This means that we need to compute the *relative scale* for every translation other than the first one.

Scaramuzza and Fraundorfer [36] proposed a possible approach for relative scale estimation. If we want to estimate the relative scale for the motion t_k, between instants $k-1$ and k, we need to compute two sets of triangulated world points X_{k-1} and X_k by using the two image pairs I_{k-2} and I_{k-1}, and I_{k-1} and I_k first. Then, given i and j, two world points that belong to both sets X_{k-1} and X_k, we can compute the relative scale accordingly to these points using:

$$r = \frac{\|X_{k-1,i} - X_{k-1,j}\|}{\|X_{k,i} - X_{k,j}\|}. \tag{2.4}$$

For each world points pair, we obtain a different values of r; we can then select

the average of r (or better, its median in case of outliers). Note that the exact *absolute scale* for the whole camera motion and environment reconstruction is still left unknown.

2.6 Bundle Adjustment

As shown in section 2.4, the error accumulation problem introduces unavoidable errors for each camera pose estimation. Therefore, a SfM pipeline has to incorporate a refinement process for both camera poses and triangulated points. This step is called *bundle adjustment* (BA) and its goal is to minimize the *reprojection error*. Point reprojection is the function that projects a world point to image space; in particular, the reprojection $\bar{\mathbf{m}}_i^j$ of the scene point $\mathbf{M}^j$ according to camera i is defined as

$$\bar{\mathbf{m}}_i^j = K_i[R_i|\mathbf{t}_i]\mathbf{M}^j.$$

Bundle adjustment can then be expressed as the following minimization problem

$$\min_{R_i,\mathbf{t}_i,\mathbf{M}^j} \sum_{i=1}^{N}\sum_{j=1}^{n} d(K_i[R_i|\mathbf{t}_i]\mathbf{M}^j, \mathbf{m}_i^j),$$

where $\mathbf{m}_i^j$ is the image point of the i-th camera corresponding to $\mathbf{M}^j$. This non linear least squares minimization is usually solved using the Levenberg Marquardt method [46, 11, 22]. The term *bundle* refers to the fact that both the camera position and the triangulated points are jointly refined. Most BA implementations keep the camera poses fixed, solve for the triangulated points, then they optimize for the poses in an alternate fashion until the desired precision is reached.

BA is a delicate step that can affect a SfM pipeline in many ways [26, 46, 11] as a poorly planned use of it can reduce computation speed and fail to obtain the desired accuracy.

2.7 Robust Estimation

In the previous sections, we have described the core steps of a SfM pipeline. Even though these are all components needed for a working SfM software, there are

some improvements widely used by similar pipelines nowadays that can further increase the reconstruction and pose estimation precision and robustness, and, in general, make these pipelines more effective. In the following sections, we review some of the most common improvements employed in SfM software.

2.7.1 Feature Detectors

As we have discussed in section 2.5.1, in order to estimate the essential matrix, we need to find corresponding points in image pairs. Typically, these points are called *feature points* or *keypoints*. A feature point is an area in an image that is likely to be unique; i.e., it is a point of interest such as a corner. Furthermore, for each feature point, we can extract some local information that describes it, this piece of information is called *feature descriptor*, and it is usually stored as a vector. There are many different algorithms to extract feature points and descriptors such as the *Moravec detector* [32], the *Harris corner detector* [9], the *SIFT* keypoints detector and descriptor [27], *SURF* keypoints detector and descriptor [3], etc. Some of these feature points or descriptors can be affected by rotations and scale. Since SfM is based on many pictures taken from different views, SIFT features have always been preferred over other less robust features because of their rotational and scale invariance. One of the main drawbacks is that SIFT is patented and can not be used freely as other detectors. SURF [3] is an alternative to SIFT that is also rotational and scale invariant; SURF is patented too and can not be used for commercial applications but there is an available implementation in the MATLAB's Computer Vision Toolbox and it can be used for research.

Even though SIFT and SURF are some of the most used feature algorithms because of their robustness and invariance properties, there is little they can do with poorly selected image pairs. Figure 2.4 shows some of the most common problems that may arise from correspondence matching. In general, the large variety of environments, each with its own characteristics, prevents a single technique to prevail over the others and this is one of the reason why a truly general-purpose SfM pipeline does not exists. For a more detailed discussion about the differences

amongst point detectors, see [38, 7].

2.7.2 Outliers Rejection

Since the motion estimation step is the core of SfM pipelines, its performance affects the overall software; in particular, motion estimation is influenced by the presence of outliers in features matching. A matching outlier is a pair of feature points that are erroneously considered corresponding points. If the motion estimation phase considered these points for its computation, the result would be unreliable and it could compromise the whole camera's path estimation because of the incremental nature of the SfM pipeline. In order to reduce this phenomenon, we can switch to a more robust feature points detector and use a RANSAC approach to remove outliers before the motion estimation step. RANSAC [5] is a model fitting algorithm for datasets that include outliers. This algorithm is iterative. At each iteration, it selects a subset of input data and computes the model based on this subset only. Then, RANSAC divides the complete dataset into a consensus set and outliers. If the number of outliers, computed thanks to a problem specific loss function, is less than a given threshold, the estimated model is considered valid.

RANSAC is now a standard step in SfM. In this case, a certain number of correspondences is selected randomly, E is computed according to the selected matches and the quality of the estimation is evaluated according to the reprojection error defined as

$$Err = \sum_{i=1}^{N} \sum_{j=1}^{n} d(K_i E \mathbf{M}^j, \mathbf{m}_i^j).$$

This error formulation is similar to equation 2.6, but now E is not decomposed in $[R_i|\mathbf{t}_i]$. The use of RANSAC before motion estimation provides a more reliable estimation even if the number of outliers is high (40-50%).

2.8 Full Spherical Cameras

In the following sections, we describe the full spherical photography's camera model (Section 2.8.1), the equirectangular image representation used to map spherical images on a 2D matrix of pixels and the problems for SfM algorithms caused by this format (Section 2.8.2). In Section 2.8.3, we give an overview of the efforts of previous studies on this topics and describe the contribution of our work to this field.

2.8.1 Spherical Camera Model

In Figure 2.5, we can see the full spherical camera model. The axes' origin is also the centre of projection. The image is formed on the unitary sphere whose centre coincides with the centre of projection. $\mathbf{m}$ is the projection on the sphere for the world point $\mathbf{M}$, thus $\mathbf{m}$ is a unitary vector that lies on the ray that goes from the centre of projection to the point $\mathbf{M}$. The image point $\mathbf{m}$ is defined by the spherical coordinate system composed of the two angles λ and ϕ, which represent, respectively, the longitude and latitude. We can use the following formula to convert the spherical coordinates to euclidean:

$$\mathbf{m} = \begin{pmatrix} \cos\phi \sin\lambda \\ -\sin\phi \\ \cos\phi \cos\lambda \end{pmatrix}. \tag{2.5}$$

Since this model is essentially different from the perspective camera's [43], the traditional procedures for pose estimation of the computer vision frameworks can not be used. In section 3.1.5, we show how we can still exploit equation 2.2 to estimate the essential matrix and use it to compute the camera motion.

2.8.2 Equirectangular Image Representation

The equirectangular image (Figure 2.6) is a 2D representation for omnidirectional images. It stores each 3D image points in a bidimensional matrix of pixels by mapping the two angular dimensions (latitude and longitude) to the two linear

dimensions of the image (height and width); latitude increases from top to bottom while longitude increases from left to right. In particular, the equirectangular mapping for full spherical images is the following:

$$\lambda = \frac{u}{W2\pi} - \pi \tag{2.6a}$$

$$\phi = \frac{\pi}{2} - \frac{v}{H\pi}, \tag{2.6b}$$

where W and H are, respectively, the image's width and height while u and v are the pixel coordinate in the equirectangular image, like in figure 2.6. In case of full spherical images, we have that $\lambda \in [-\pi; \pi]$ and $\phi \in [-\frac{pi}{2}; \frac{pi}{2}]$.

The equirectangular representation introduces significant distortions, especially around the poles; this compromises the robustness of the feature matching procedure which can then return incorrect estimations for the essential matrix. Similarly to the feature matching issue, distortions causes particular problems with the block-matching algorithms for disparity map computation. In Section 3.2 we describe the modified block-matching procedure we designed to reduce the effect of distortions when computing disparity maps.

2.8.3 Previous Work on Spherical SfM

As we pointed out in the previous sections, there have been few efforts to use full spherical cameras in SfM and most of the work has targeted perspective cameras. Most of the work about panoramic cameras have employed catadioptric or dioptric devices with a single image sensor (no stitching), therefore, as we said in section 2.3, none of this types of camera is capable to produce full spherical images.

Li [23, 24] developed a rectification procedure for wide FOV cameras that enables the computation of disparity maps with standard block-matching algorithms; the rig he used for testing was composed of fisheye cameras that could not provide full spherical views. Ma et al. [28] used equirectangular images produced with a Ricoh Theta S to create disparity maps. They manually selected some correspondences in order to compute the essential matrix correctly. Then, they used the epipolar geometry to remove correspondences outliers after an automatic

feature matching step. Due to the user intervention in the pipeline, they could not test their results on long sequences. Moreover, the computed disparity was noisy. On the contrary, Arican et al. [2] developed a new algorithm based on graph-cut to compute high-quality disparities of omnidirectional images directly. Im et al. [15] created a framework focused on the production of depth maps using full spherical consumer cameras in which the input was a short video sequence with small camera motion. Aly et al. [1] extracted interest points' directions from the equirectangular images using Equation 2.5, then they used the epipolar equation (2.2) to estimate the essential matrix. Their work focused only on pose estimation from a unordered set of images and they assumed the camera movements were constrained to a plane. Kangni et al. [18] converted the full spherical image format to six separate perspective projections onto the faces of a cube around the viewer. Then, they used the epipolar equation to obtain the essential matrix focusing on motion estimation without dense reconstruction. Pagani et al. [35] designed a SfM pipeline for full spherical cameras, which exploits spherical coordinates directly (without reprojecting the 3D directions on a planar image). Their work has the main goal to study the effect of the different error metrics for pose estimation. Furhtermore, they introduced a novel sparse reconstruction and pose estimation algorithm.

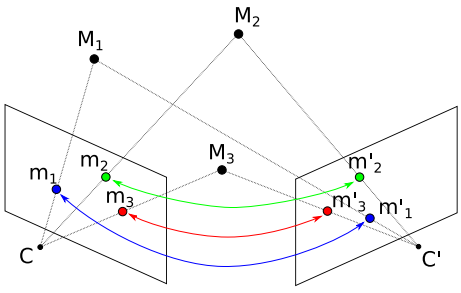
In this work, we propose a complete SfM pipeline with dense point cloud reconstruction for 360°video sequences with the following characteristics:

- no view selection is needed; i.e., it works with every frame of the input video sequence by automatically discarding the unnecessary images;
- the pose estimation is based on a 2D-to-2D approach (see section 2.5.1), and it uses both frontal and rear hemisphere's correspondences;
- it employs a novel block-based disparity estimation algorithm that reduces matching issues caused by the typical distortions of equirectangular images;
- it provides a dense point cloud as an estimation of the environment.

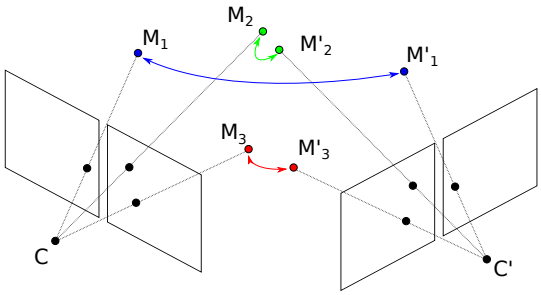
Our contributions include:

- a novel method for selecting frames to be employed for motion estimation among all frames in a video;
- a novel adaptive block-matching algorithm for disparity maps for equirectangular images;
- a working MATLAB implementation of our SfM pipeline.

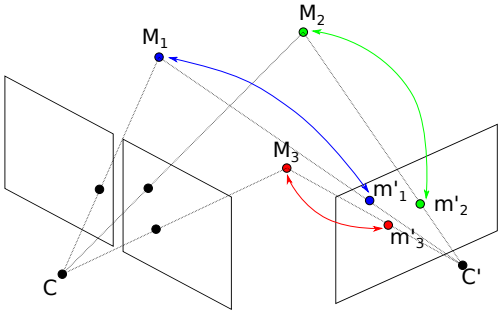
.



(a) 2D-to-2D.

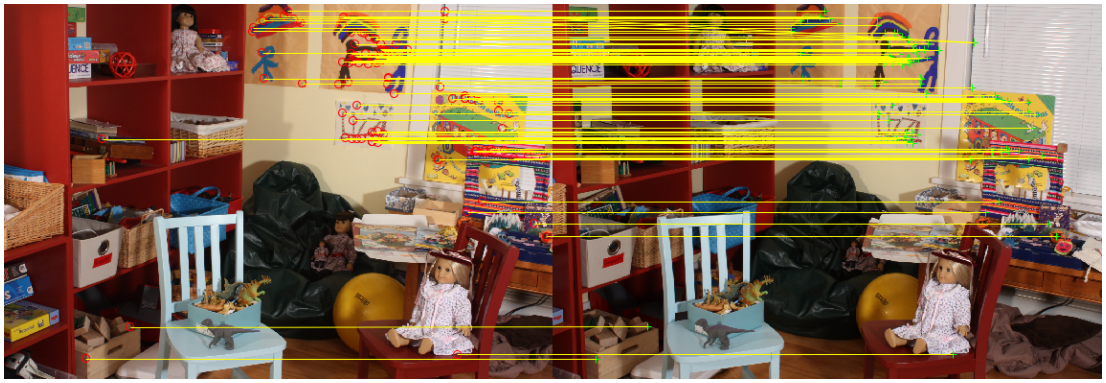


(b) 3D-to-3D.



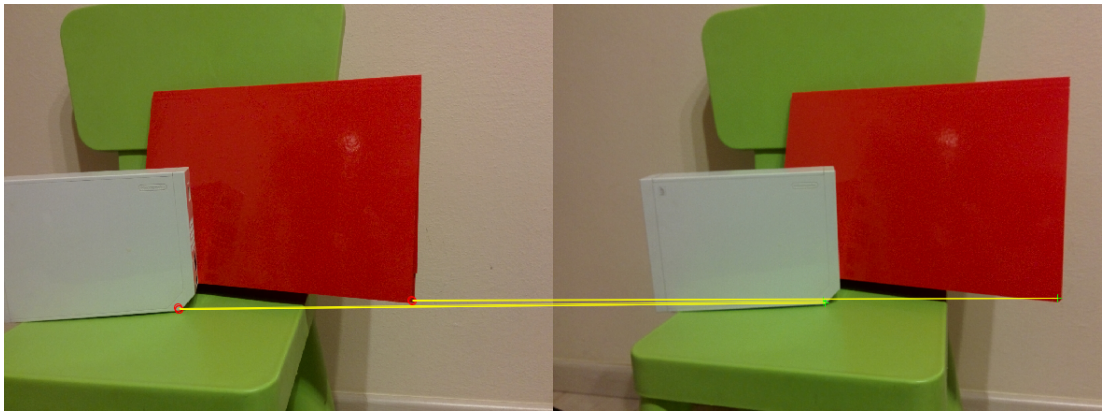
(c) 3D-to-2D.

Figure 2.3: Representation of the feature matching step for motion estimation. Uppercase letters indicates 3D points while lowercase letters are 2D image points; the new pose is C'. Note the stereo rigs in (b) and (c) to highlight the need for triangulated world points with these approaches.



(a) A subset of the matches found in this image pair.

(b) Motion blur reduces both the number and quality of matches.



(c) Textureless images provide few feature points.

Figure 2.4: Some examples of problems that may occur when matching correspondence search.

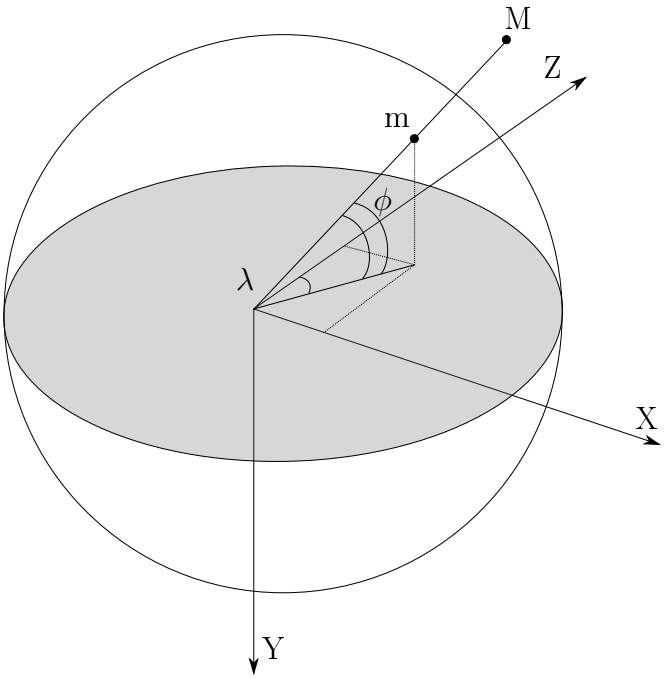


Figure 2.5: The full spherical camera model.

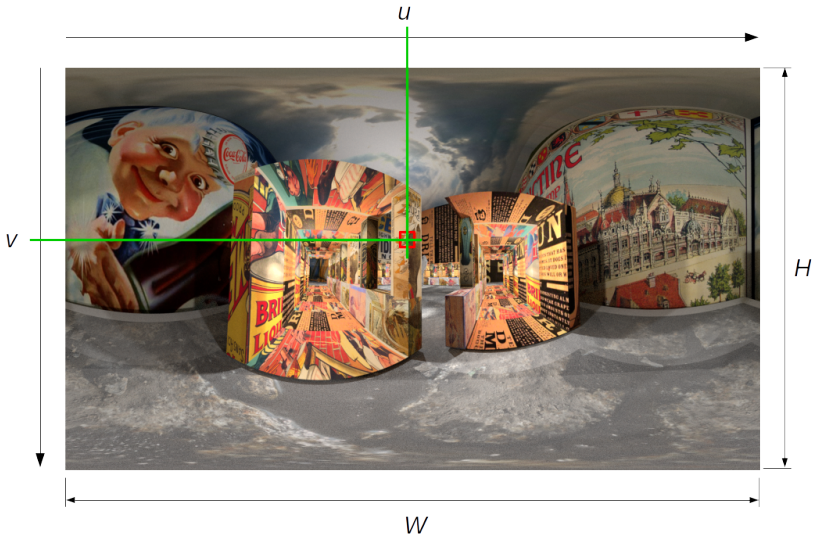


Figure 2.6: The equirectangular image representation maps the 3D image points in spherical coordinates to the 2D image point (u, v).

Chapter 3

Spherical SfM

In this chapter, we describe the pipeline that we designed to create a dense point cloud from a set of equirectangular images. In Section 3.1, we describe the first phase of our pipeline: the selection of the frames from the input sequence and the estimatation of the camera trajectory. In Section 3.2, we describe our densification algorithm for equirectangular images. Fig. 3.1 shows a coarse visualization of these two macro parts of the pipeline. Pose estimation and multiple view stereo reconstruction are expanded further in Figures 3.2 and 3.4.

3.1 Pose Estimation

The cameras' poses estimation phase is similar to the classical visual odometry pipeline. The main differences lie in the variables' format and in the details concerning the procedures used. For each new frame (in the equirectangular representation), we locate the SURF keypoints [3]; we consider those keypoints whose inclination angle is in the range $[-60°; +60°]$. This is because the poles may be affected by large distortions, thus robust matches outside this interval are rare. Then, we look for matches in the last two frames. A filter performs a statistical analysis on the correspondences found and decides whether the frame is suitable for a robust pose estimation or not (in this case the frame is discarded).

If the frame is kept, its matches with the previous ones are converted and used for the essential matrix (E) estimation. Once E is computed, it can be

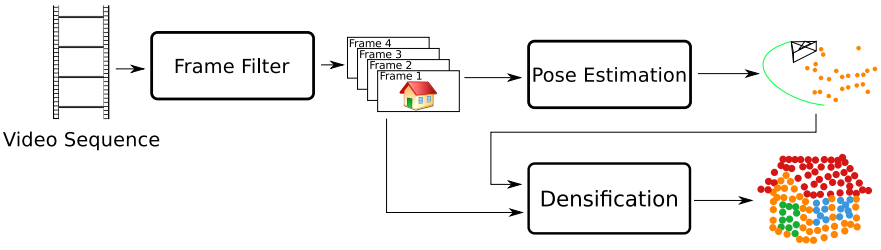


Figure 3.1: The proposed SfM pipeline for spherical images: the frame selector chooses which frames are relevant for the next processing steps; the pose estimation phase returns the cameras' poses and the sparse points cloud; the densification step uses the input images and the previous camera's poses to generate a dense reconstruction of the environment. See Figures 3.2 and 3.4.

decomposed into the $[R|t]$ form; where R is a rotation matrix and $\mathbf{t}$ is a translation vector up to an unknown scale factor.

The relative scale can be estimated by using world points that have been triangulated through matches and the frame considered in the previous pipeline iteration.

Then, the pose of each camera can be described as the composition of the motions between each view pair. We perform a bundle adjustment for the last five poses in order to reduce the effect of drift (see Section 2.4 for details about drift).

After the processing of all images, a final bundle adjustment step is performed. This optimizes every camera pose (previously estimated) in order to reduce drift error further.

Figure 2.5 shows the coordinate system used in our pipeline: the X-axis points right, the Y-axis points down, and the Z-axis points forward. We consider the spherical image divided into two hemispheres: front and back. We call *frontal points* the image points such that the z component is positive, otherwise we use the term *rear points*. When we need to express spherical coordinates, we use the same angles as shown in Figure 2.5: the angle between the Z-axis and the projection of m on the XZ-plane in the clockwise direction is the longitude angle (λ), and the angle between the negative Y-axis and the same projection of m is

the latitude angle (ϕ).

3.1.1 Keypoints and Features Extraction

In Section 3.1, we have introduced the need for features detection in order to find correspondences among the image pairs. In our pipeline, we use SURF features and descriptors [3], which is a blob detector partly inspired by SIFT [27]. SURF performs convolution on integral images with box filters in order to compute the Hessian matrix in scale space. This descriptor exploit keypoint's neighbourhood response to the Haar wavelet. SURF, as SIFT, is patented. However, it can be used freely in non-commercial applications and for academic research. An implementation of the SURF detector is available in the MATLAB's Computer Vision Toolbox, and we employed it in our pipeline.

3.1.2 Keypoints Filtering

The equirectangular representation for spherical images introduces significant distortions around poles. Keypoints in those areas are unlikely to match reliably with other points in a consecutive view, therefore they are discarded. Besides, in our experiments setup, the North and South poles typically point upward and downward respectively, while most of the matches useful for pose estimation comes from the sides of the camera. Therefore, the removal of interesting points near the poles does not affect the final result.

3.1.3 Features Matching

The matching strategy, which we carried out for equirectangular images, is the same one adopted with standard images: two keypoints matches if the distance between their descriptors is less than a given threshold (see Section 2.5.1 for the 2D-to-2D features matching approach). Ambiguous matches are discarded if the ratio between the distances to the two closest matches is above a maximum. We enforce matching robustness by exploiting the fact that the correspondences are unique; i.e., only one feature in the first image can match with another one in the

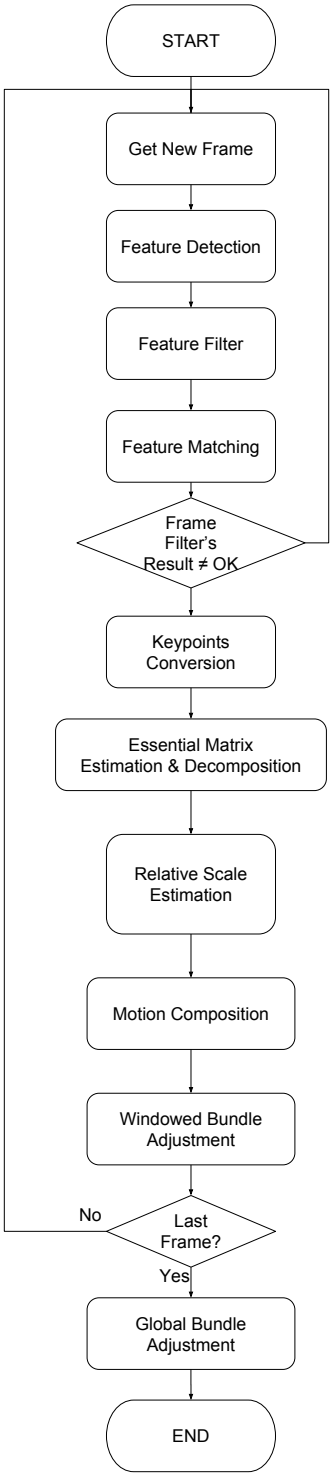


Figure 3.2: The flow chart of steps performed in the pose estimation phase of our SfM pipeline.

second image. This is achieved with two passes of the matching procedure. During the first pass, each feature in the first images is checked for correspondences in the second image. In the second pass, each feature of the second image that has at least one correspondence in the first image is checked to see what interesting point of the first image it corresponds to and the match with the highest confidence is kept.

3.1.4 Frame Filter

In this step of the pipeline, matches are analyzed in order to decide whether a frame has to be kept because it is useful for motion estimation or not. We assume that the scene to capture is static; i.e., no moving objects such as cars or people. In this case, the apparent movement of corresponding objects in different views is caused by the camera motion only. The filter computes the disparity between every match found in two views; i.e., it compares the median of the 20% of matches with the largest angular displacements with a threshold. If such median is above the threshold, the frame is kept for further processing in the pipeline, otherwise it is discarded. The reason because we consider the correspondences with the greatest apparent movement is because, when the camera rotation is limited, the points that move very little in consecutive views are typically far. The selection of these points for motion estimation can be counter-productive because they can easily produce numerical errors. The threshold value depends on the specific environment: if the scene is large and most of the surfaces are far from the camera, a slightly reduced value may improve the estimation quality, while, if the environment is smaller, we may benefit from an increased threshold value. We experimentally choose threshold value of 5 radians for real environments while we increase it up to 10 radians for computer-generated scenes (see Chapter 4 for details about the experiments).

3.1.5 Keypoints Conversion

This step converts the keypoint format of equirectangular images to a new one, which is suitable for estimating E. First, the latitude and longitude coordinates of each feature point are extracted from the 2D mapping according to Equation 2.6. This equation returns spherical coordinates for each feature point. Then, we can convert each of them to its cartesian format using Equation 2.5. In order to estimate E for each view pair, we use Equation 2.2, which was introduced by Longuet-Higgins [25]. The point coordinates, which we obtain from Equation 2.6 and Equation 2.5, do not need normalisation because there is no intrinsic parameter for the full spherical camera model.

In order to exploit the functions to estimate E available in the MATLAB's Computer Vision Toolbox, we need to perform an additional step, because this toolbox's routines deal with standard 2D perspective images. Therefore, they expect 2D points since the input arguments are image points. However, our camera provides image points on a sphere, and so are 3D points. To account for this problem we point out that we can multiply the points $\mathbf{p}$ and $\mathbf{p}'$ by two scalars λ and λ' and the equation is still valid. Thus, we obtain

$$\lambda' \mathbf{p}'^{\top} E \lambda \mathbf{p} = 0.$$

Therefore, we divide the 3D points obtained from the spherical images by their 3rd component, discard it, and use the resulting 2D points as input for the MATLAB's function `estimateEssentialMatrix`, which estimates E.

Division by 3rd component vs. Projection

Kangni et al. [18] converted the feature points extracted from spherical images to their projected images on a planar surface and used traditional techniques to estimate the camera poses. Even though our method is very similar to Kangni et al., there are the following differences:

- Projection has to take into account all the parameters of the perspective camera model [43];

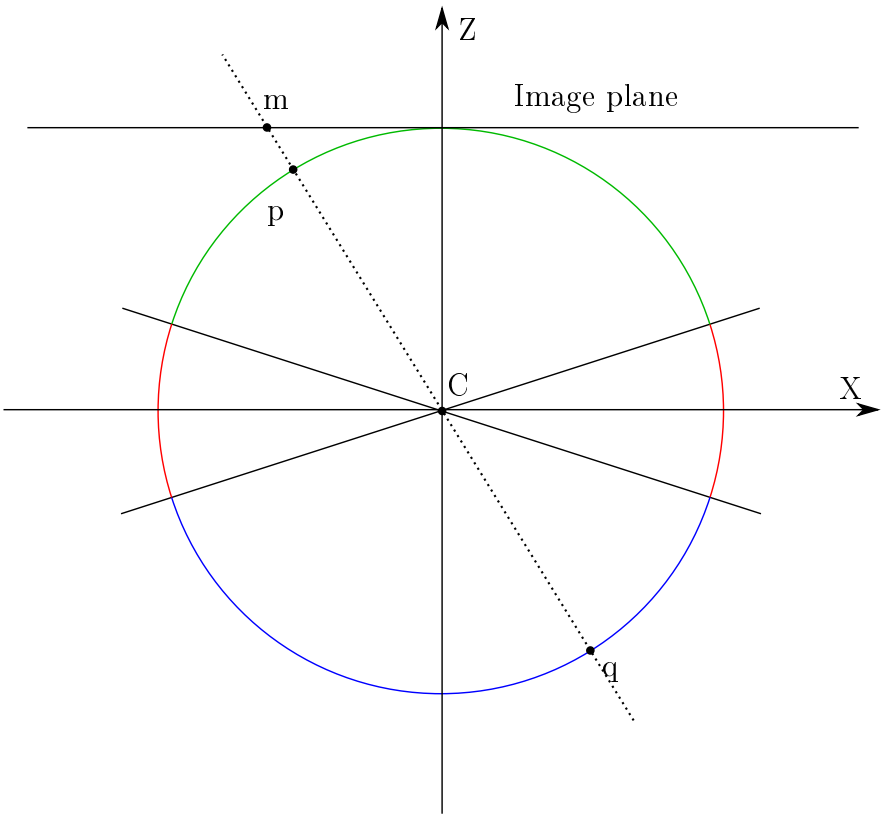


Figure 3.3: A top view representation of the full spherical image's different portions. The feature points that lie on the blue and green parts of the sphere are kept and, after the conversion described in Section 3.1.5, they are used for E estimation. On the other hand, the points that lie on the red portions of the sphere are just discarded. Both the frontal point p and the rear point q are projected in m.

- two separate projections are needed if we want to use both frontal and rear points.

Projecting a point means that we have to deal with perspective geometry and its parameters (e.g., pixel size or density), the principal point coordinates, focal length, image size, etc.

We can think to our method as a simplified perspective projection in which f_x and f_y are both set to 1, and u_0 and v_0 are 0. The main difference between our method and a standard projection is that, if we just divide each feature point by its z-coordinate, we do not need to differentiate between frontal and rear points. Figure 3.3 shows how both the frontal point, p, and the rear one, q, are projected to the same point m on the image plane. This is not an issue, on the contrary, this helps the estimation of E because it adds more redundant data to the input.

Note that we need to take care of numerical errors only: if we divide by a small number, the result is affected by a large error. Therefore, we set a minimum value for the absolute value of the z-coordinate a feature point must have in order to be used in motion estimation. If the z of a feature point is below the threshold, we discard the point. We called this threshold parameter z_{min}. In Figure 3.3, the red area is not taken into account during motion estimation because the value of the magnitude of the z-coordinate of points there is below z_{min}. Selecting points whose last component is above a certain threshold is equivalent to discarding those points that do not fit in the image plane when projecting them. More details on this theory can be found in the Szeliski's book [43] and Hartley and Zisserman's books [11].

3.1.6 Essential Matrix Estimation and Decomposition

As we have described in the previous section, E is estimated by the function `estimateEssentialMatrix` of the MATLAB's Computer Vision Toolbox. We use both frontal and rear image points for this estimation because more correspondences between image pairs produces more accurate results.

Once E is estimated, we need to decompose it in the $[R|\mathbf{t}]$ form. This is again

performed by a Computer Vision Toolbox's function; i.e., `relativeCameraPose`. The inputs for this function are E, the camera parameters, and matches found in the last two images. We need these matches because there are four possible decompositions for E. Each of these represents a different physical configuration for the cameras and world points. In order to decide which decomposition is correct, in general, it is necessary to reproject the matches in their corresponding world points according to each decomposition and selects the one that reprojects most of the correspondences in front of both cameras. Since our cameras are full spherical and the image points belong to the rear hemisphere too, we need to pay attention to use the frontal points only to resolve this ambiguity. The reduced number of matching points for the input of the `relativeCameraPose` function does not compromise the accuracy of the pose estimated, since those matches are used only to choose the correct decomposition.

3.1.7 Relative Scale Estimation

Every relative motion between two views can only be estimated up to an unknown scale factor. Indeed the scale affects just the translation, but it has to be computed in order to create a coherent set of camera poses. We obtain the relative scale using Equation 2.4 [36]. This equation provides as many results as 3D points present in the last three frames. In order to compensate for the effects of outliers, we take the median.

3.1.8 Motion Composition

Once we have estimated the relative motion between two views, we use Equation 2.1 to compute the new camera pose C_n from the last estimated pose C_{n-1} and the results R_n and t_n obtained from the decomposition of E. We set the first orientation, R_0, to I and the magnitude of the first translation, t_0, to 1.

3.1.9 Windowed Bundle Adjustment

Since every local motion estimation is inevitably affected by error, the overall camera's path estimation tends to deviate from the real trajectory. In order to reduce the drift and to get closer to a better starting point for the final bundle adjustment, we perform a bundle adjustment over the last five poses every time we process a new frame. This local optimization step on the most recent subset of frames is called *Windowed Bundle Adjustment*.

The bundle adjustment tries to reduce the sum of reprojection errors by changing the world points, and camera positions. We keep the camera poses associated with the two oldest frames of the window fixed in order to prevent the adjustment from modifying the reconstruction's scale. This constraint also helps reducing the number of variables for the adjustment.

3.1.10 Global Bundle Adjustment

When all the views have been processed, we perform a final bundle adjustment step, in order to further reduce drift. Again, the first two poses are fixed in order to keep the relative scale.

3.2 Point Cloud Densification

In this chapter, we describe our pipeline's multi-view reconstruction step. Its goal is to create a dense point cloud from the sparse one that we obtain from the previous pose estimation step. As we said in Section 2.2, the approach we followed for the reconstruction is based on merging several depth maps that have been computed from a set of consecutive image pairs. In order to compute the disparity maps, we have to transform each image pair through the *rectification* transformation, i.e. the images have to be transformed as they were taken from cameras whose X-axes were aligned along the baseline and whose Z-axes lay on the same plane. The purpose of the rectification is to speed up the computation and reduce the complexity of the disparity estimation step; it also helps by re-

ducing the wrong correspondences we would find with non-rectified image pairs. Section 3.2.1 describes the rectification for spherical cameras. Once we have rectified image pairs, we use a block-matching algorithm to create the disparity map. In Section 3.2.2, we describe the specific correspondence metric we use and the details about our own solution to reduce the effect of the equirectangular image representation distortions (Section 3.2.2). Sections 3.2.3 and 3.2.4 describe how we compute the 3D points' coordinates and express these points in a unique coordinate system. Figure 3.4 shows all the steps needed to create a dense point cloud from the previously estimated camera poses and corresponding views. In this thesis, we use the naming convention shown in Figure 3.5.

3.2.1 Image Pairs Rectification

The goal of the rectification is to transform the two images of a pair as they were taken with cameras whose x-axes were aligned along the same line and both pointing toward the same direction, and whose z-axes were parallel. With this peculiar cameras' arrangement and in case of perspective cameras, corresponding points appears on the same row in the two rectified images, thus the correspondences search domain is shrunk from a 2D-space to a 1D-space. In case of spherical cameras the rectification aligns corresponding points on the same meridian, hence the same column in the two equirectangular images. It is important to notice that, given a point in the reference image L, the possible positions that the corresponding point in the other image can assume are always constrained to a line, even if the pair is not rectified; these lines are called *epipolar lines* [11, 43]. Figure 3.6 shows the same image pair before and after rectification.

The problem is that, without rectification, the points that lie on a specific epipolar line, have different coordinates depending on where they are on the image sphere and, in general, they do not share the same value for any of their latitude or longitude angles. Thus the epipolar lines do not map to a line in the equirectangular image, i.e. they are not represented by vertical or horizontal lines in this bidimensional mapping of the spherical image. On the other hand, if the two images are rectified, all the points that lie on an epipolar line have the

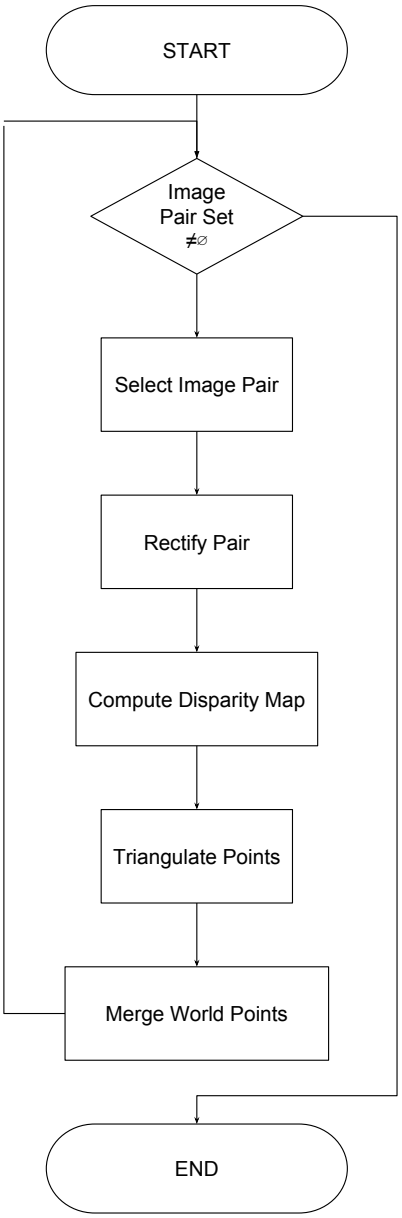


Figure 3.4: The flow chart representing the whole reconstruction phase.

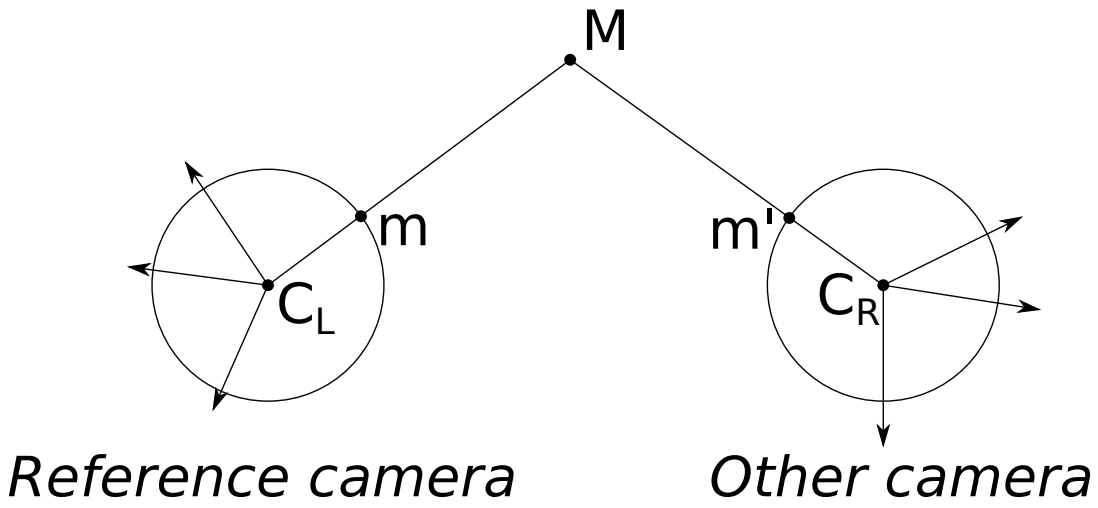


Figure 3.5: The naming convention used in this thesis: left camera, C_L, is the reference camera in an image pair; point M is the *world point* (see Section 3.2.3), and m and m' are the two image points with respect to the reference and other camera respectively.

Figure 3.6: An example of the same image pair before (left) and after rectification (right). Notice how, on the left of all the images, the corner of the letter "e" is aligned in the right hand pair while it is not in the left one.

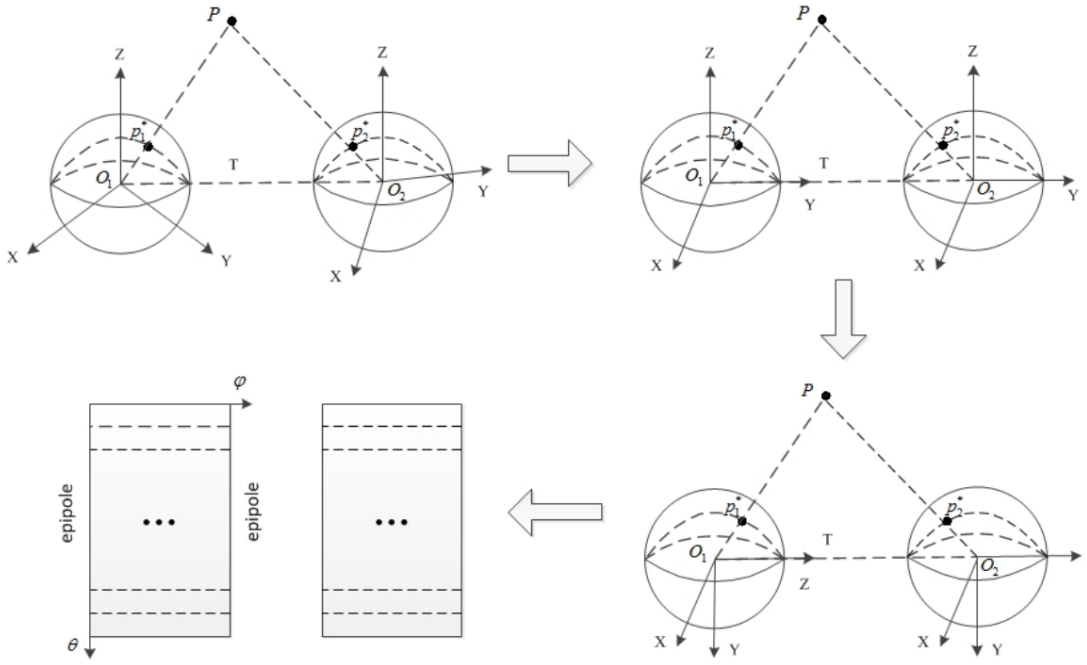


Figure 3.7: This figure shows the transformation performed with each rotation. First the two cameras are rotated such that their X-axes become aligned and their Z-axes are parallel, then a 90°rotation about the Z-axes aligns the cameras' meridians to the epipolar lines. The resulting equirectangular images have corresponding points that lie on the same column. This picture is taken from the work of Ma et al. [28].

same value for their longitude component, thus they have the same horizontal coordinate in the equirectangular image.

We use the same rectification used by Ma et al. in [28] and obtain epipolar lines that map to columns in the equirectangular image. In particular, given an image pair, the rectification consists in a specific rotation for each of the two cameras. Let's call R_L and R_R the rotation applied to the left and right camera respectively; each rotation is obtained by the composition of other transformations. Figure 3.7 shows the sequence of rotations we combine to obtain R_L and R_R. First the two rotations $R_{L,1}$ and $R_{R,1}$ align the two cameras' Y-axes such that they are lie on the baseline, both pointing in the same direction. Then $R_{R,2}$ rotates camera R around its Y-axes so that the orientation of R is the same as L's. We apply one last rotation, R_3 to both cameras to make them lie on a side, thus aligning the

epipolar lines to the image spheres' meridians. Hence the overall rotations R_L and R_R that we must apply to cameras L and R respectively are:

$$R_L = R_{L,1} R_3 \tag{3.1}$$

$$R_R = R_{R,1} R_{R,2} R_3, \tag{3.2}$$

where all the rotations are expressed in the post-multiplication form.

3.2.2 Disparity Map Computation

Given an image pair, the disparity map stores the distance in pixel between a point found in the reference image L and its correspondence in the other image R. In case of rectified equirectangular images the disparity of a specific pixel refers to the difference between the v-coordinate of the point in L and the same coordinate in R. Contrary to feature matching (Section 3.1.3), we do not want to find the distance for few keypoints; in fact, the aim of this step is to find the correspondence for each point that appears in the reference image. The block-matching class of algorithms computes disparities by extracting the pixels surrounding a point that it wants to match and computing a similarity or error metric with the neighbourhood pixels of each of the candidate points in the other image. The small squared area extracted from each point is called *patch* or *block*, thus the name block-matching. There are several parameters in block-matching algorithms that affect the resulting disparity maps; among them, there are:

- *patch size*: the number of pixels for each patch's side (N);
- *maximum disparity*: this defines the boundaries for the search, i.e. how far, in pixels, the algorithm moves above and below the corresponding pixel in the other image (in the perspective case, this value refers to the horizontal distance traveled by the searching procedure);
- similarity/error metric employed: how the error or similarity metric is calculated.

We implemented both SSD and NCC metric for our block-matching algorithm. Given the two images I_L and I_R, the SSD metric is computed as:

$$SSD(u,v,d) = \sum_{(k,l)} (I_L(u+k, v+l) - I_R(u+k, v+l+d))^2,$$

where $k, l \in [-N, N]$ and $I_X(i,j)$ is the intensity level of pixel (i,j) in the gray image X. In order to increase the matching robustness, we minimize the result of the interpolation between the SSD response to both the original and differential images according to a parameter α [31]. Then, our SSD result is given by

$$\begin{aligned} SSD(u,v,d) = (1-\alpha) &\left[\sum_{(k,l)} (I_L(u+k, v+l) - I_R(u+k, v+l+d))^2 \right] + \\ +\alpha &\left[\sum_{(k,l)} (I_{x,L}(u+k, v+l) - I_{x,R}(u+k, v+l+d))^2 + \right. \\ &\left. + \sum_{(k,l)} (I_{y,L}(u+k, v+l) - I_{y,R}(u+k, v+l+d))^2 \right], \end{aligned}$$

where $I_{x,J}$ and $I_{y,J}$ are the differential results of image J computed along the X and Y direction respectively. Our adaptive block-matching algorithm can also employ the similarity metric NCC that is computed as:

$$NCC(v_0, v_1) = \frac{\sum_{j=1}^{N^2} v_0(j) \cdot v_1(j)^2}{\sqrt{\sum_{j=1}^{N^2} v_0(j) \sum_{j=1}^{N^2} v_1(j)}},$$

where $v1(i)$ and $v1(i)$ are the i-th pixel of the reference and test patch respectively. The patch size is chosen experimentally and varies with the particular scene we use; it is usually set to 5, 7 or 9 pixels for our 480x270 images (we resize the images after the rectification step in order to reduce the time needed to compute the disparity maps). The maximum disparity is computed automatically for each pair; in particular, we look for feature matches in the two images and compute the maximum vertical distance between matches, then we use such a value as maximum disparity.

Since the disparity map is based on the rectification procedure, its quality is deeply affected by the previous step's result, and rectification, in turn, depends on the accuracy of the estimated poses. In order to compensate for rectification

errors, we try to match each point in the reference image with points that lie on a vertical stripe instead of a single-pixel column in the other image when we look for correspondences during the disparity map computation. With this simple modification, we can compute disparity maps even in presence of small rectification errors. We set the stripes' width to 3 pixels.

Yet another problem that arises when computing disparity maps is *occlusion*: objects that are far from the viewer may be covered by a foreground object in one of the two views. Computing the disparity of a point in the reference image that is occluded in the other image would produce an inevitably wrong result. To remove occluded points from the disparity map, we perform the additional *cross-checking* test [31]. Our cross-checking step computes two disparity maps, $M_{L,R}$ and $M_{R,L}$, for each image pair. Each map considers a different image as reference; in particular, $M_{X,Y}$ considers image I_X as the reference and I_Y as the other image. Let's assume $M_{L,R}(i,j) = d$, then we keep only those points that satisfy the condition:

$$i + M_{R,L}(i + d, j) + d < t,$$

where t is a threshold in pixels that we set to 2. Occluded points are not likely to verify the constraint above, hence they are discarded.

Patch Creation

The disparity map computing procedure that we have described so far is not different from an equivalent procedure for the perspective case, with the only exception of the orientation of the epipolar lines in the rectified pairs (that are vertical instead of horizontal). The main difference of our reconstruction phase from the traditional ones is in the patch creation step. The patch is usually obtained by extracting the neighbourhood pixels of the point we want to match, but, if we apply this method to equirectangular images, the patch retains all the distortions introduced by the particular image format, and these can prevent correct matching results. Instead, we introduce a novel patch creation algorithm that removes such distortions by projecting the pixels around the points on a plane. In this way, we obtain a less distorted patch that we can then use in the

disparity map routine we described in the previous section. This is especially useful when the points are close to highly distorted areas, like the ones around the poles.

In Figure 3.8 we show the geometry involved in the patch creation step. In order to create a patch, we need the actual patch size with respect to the image sphere radius. To obtain the patch size we set the *patch resolution*, N, that is the length in pixels of each side of the square patch. Then we compute the two angular resolutions of the equirectangular image with the formulas:

$$\sigma_u = \frac{2\pi}{W} \tag{3.3a}$$

$$\sigma_v = \frac{\pi}{H}, \tag{3.3b}$$

where W and H are the with and height of the equirectangular image, then we set $\sigma_{max} = max(\sigma_u, \sigma_v)$. Given the patch resolution and the image's angular density, σ_{max}, we can compute the actual patch size, s, accordingly to the formula

$$s = 2 \tan\left(\frac{N\sigma_{max}}{2}\right)$$

Once we have the patch size and its resolution, we project the point of the spherical image on the patch itself. When we have to compute the patch that surrounds an equirectangular image point whose coordinates are (λ, ϕ), we position the patch's plane such that it is tangent to the image sphere in (λ, ϕ), then we project the point's neighbourhood pixels on the patch.

3.2.3 World Points Triangulation

Once we have the disparity map for a given image pair, we analyse each disparity and compute a 3D point, the so-called *world point*, for each non-occluded point in the disparity map. In fact, we use the disparity to obtain the image coordinates (latitude and longitude) of the same world point as it appears in the two camera views, thus we have the two directions where this point lies; the world point is the intersection of these directions. In practice, the lines defined by these two directions do not intersect because of errors introduced by several factors (lens distortions, image resolution, etc.). The solution is to find the mean point of

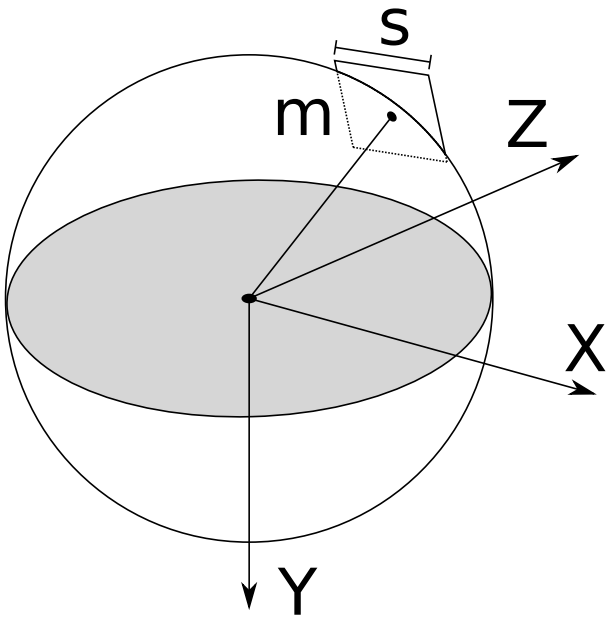


Figure 3.8: Patch creation: the neighbourhood pixels of the image point m are projected on the patch. This approach prevents the patch from containing the same distortions that are present in the equirectanguar image format.

the distance between the directions and consider it as the best estimation of the real world point's position. Our triangulating function exploit the MATLAB's `triangulateMidPoint` that implements the solution proposed by Hartley and Sturm in [12]. Since our matches are located on the two image spheres, they are 3D points, thus we have to divide each of them by their 3rd component and discard it before passing them as parameters to the MATLAB's function. This step is exactly the same that we used in order to prepare the input parameters for the essential matrix estimation function that we call in the camera pose estimation phase. The same considerations we made in Section 3.1.5 hold and, in particular, we do not consider those points whose 3rd component magnitude is less than a given threshold, $z_{min} = 0.01$.

The triangulated points are expressed in a reference system that has the same orientation and position of the reference camera of the view pair. Since the image coordinates of the matches are given for the rectified pairs, we have to apply the inverse rotation, $R_L^\top$, in order to obtain the true world point positions

(see Section 3.2.1).

3.2.4 Merging World Points

As we pointed out in the previous section, the world points are always expressed in a reference system whose centre coincides with the reference camera's centre of projection and that is oriented as the reference camera itself. Thus, in order to obtain a coherent point cloud, we have to express every world point in a global coordinate system. We want to merge all the point clouds obtained by the image pairs according to the coordinate system defined by the first view analysed. Hence, for each world point $\mathbf{M}$ obtained by an image pair whose reference camera is located at $\mathbf{t}$ with orientation R (in the post-multiplication form), we have to compute the translated point $\mathbf{M}'$ as

$$\mathbf{M}' = \mathbf{M}R + \mathbf{t}.$$

Once every point has been translated, the overall point cloud is simply the union of every set of image pairs' world points.

Chapter 4

Results

In this chapter, we comment the results obtained with our SfM pipeline. We tested our approach with both computer-generated and real-world environments. We compare the pose estimation results with ground truth in the synthetic cases while we only give a qualitative evaluation for the multi-view stereo reconstruction step for both the synthetic and real video sequences. We implemented our pipeline in MATLAB R2017a, and we collected our results using a computer equipped with an Intel i5-2400 at 3.10 GHz and 12 GB of RAM.

4.1 Datasets Description

4.1.1 Synthetic Scenes

The first computer-generated environment is a 3D simple scene that we created in Blender 2.78c. This scene is composed of a room with 5 polyhedrons. We added some image textures to both room's walls and shapes'faces to increase matching inliers. The camera moves on a continuous curve around the polyhedrons and both its position and orientation change. Figure 4.1 shows some rendered images for this environment.

In the second computer-generated scene, we recreated a town square surrounded by a covered walkway (i.e., *loggia*). The roof of the walkway is supported by columns on the inner side, the one that is oriented toward the square's

Figure 4.1: The first computer-generated environment that we used in our tests and an example equirectangular image from this scene: two views of the scene (top row), wireframe visualizations of the same scene (middle row), and an example equirectangular image produced with this setup (bottom).

centre, and by a wall on the outer side. Again the environment is contained in a room and every surface is textured. The camera moves along the walkway with a non-uniform speed while pointing toward the centre of the square. Figure 4.2 shows some images for this second environment.

4.1.2 Real environment

We tested our pipeline with a real video footage too. In this case, we do not have the ground truth for the camera poses. However, the quality of the reconstruction is a good indicator of the pipeline performance. We captured the real-world sequence with a Ricoh Theta S camera in a town square surrounded by a *loggia*. As before, we walked along the covered walks around the square, keeping the camera above the user's head using a stick. The user does not influence the camera poses estimation because it appears in the pole region of the image sphere and, as we said in Section 3.1, we discard potential matches in these areas. We present some pictures of the real-world environment together with some examples of the equirectangular images of the town square in Figure 4.3.

4.2 Experiment Results

4.2.1 Window Size Test

We performed an experiment to choose the optimal window size for the proposed windowed bundle adjustment (Section 3.1.9). In this experiment, we used the synthetic town square and computed the sum of absolute pose error. In particular, we consider the location and orientation errors for the estimated poses. Figure 4.4 shows the comparison of the resulted location error for 20 estimated poses with respect to several windowed and non-windowed adjustment techniques. In particular, we can see that the pose error is minimal when we employ the windowed bundle adjustment with a window that contains five poses at a time. Windows smaller than five do not bring substantial improvements (as we pointed out in Section 3.1.9; to maintain the proper scale, we keep the two poses in a win-

Figure 4.2: The computer-generated town square environment: two views of the model (top row) and two examples of equirectangular images produced from this scene (middle and bottom).

(a) Panoramic view of the real town square.

(b) Example frame from the real town square sequence.

Figure 4.3: The town square we used in our real-world reconstruction test.

dow fixed), while the error also increases for larger windows. The performance deterioration when using larger windows is due to the fact that the windowed adjustment may suffer from the high number of views to be optimized: if a series of views with wrong estimated poses are already inside the window and a new pose enters it, the prevalent number of low-quality estimation may result in the last pose to be aligned to the wrong trajectory of the previous estimations. Due to this phenomenon, all next views may be incorrectly optimized. This reduces the performance of the overall pose estimation.

4.2.2 Camera Motion Estimation

In this test, we analyzed the overall results of the initial step of our pipeline, the motion estimation phase. We used the 140 rendered frames of our synthetic town square environment. With the threshold set to 10 radians, the frame filter

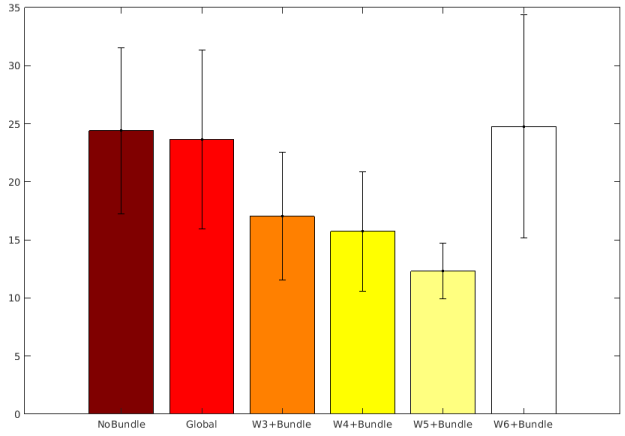


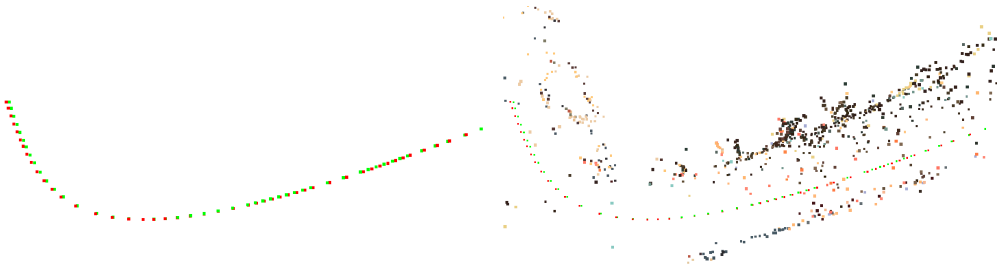
Figure 4.4: Location error with respect to different global refinement techniques. The proposed windowed bundle adjustment with a window of size 5 gives the best performance. This choice is optimal also for the orientation error.

selected 48 frames to be used for motion estimation. Figure 4.5 shows the actual and estimated trajectory of the camera in the synthetic environment. As we can see in the figure, the estimated motion is accurate even if many frames were discarded by the frame filter. This demonstrates that the frame filter is able to select a good subset of non-redundant frames.

We ran our pipeline with the real town square video too; in this case, the frame filter selected 48 frames out of 200. The estimated trajectory is shown in Figure 4.8. We do not have the trajectory ground truth for the real scenario but, as we can see in Figure 4.8 (right) and Figure 4.9, the overall reconstruction of the surrounding is correct: all the separate set of world points from each image pair merged together without any evident misalignment problem and the dense point cloud is a good approximation of the scene. These reconstruction characteristics are the results of a correct estimated trajectory.

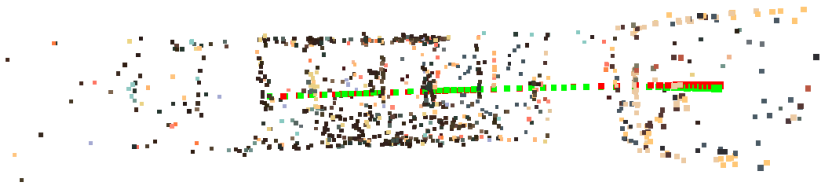
4.2.3 Disparity Maps

Figures 4.6 and 4.7 show some examples of disparity maps computed with our adaptive block-matching algorithm and the comparison with a traditional block-matching algorithm. Darker colours refer to the points with less disparity thus,



(a) Camera's trajectory (green) with ground truth (red).

(b) Camera's trajectory and sparse point cloud.



(c) Another view of the sparse point cloud.

Figure 4.5: The results of the pose estimation phase: estimated poses (green points) compard with ground truth (red points) (a); the sparse point cloud composed of the triangulated features points (b); another view of the sparse point cloud where we can distinguish the columns of the inner side of the walkway (c).

farther from the camera. On the other hand, the closer objects have brighter colours.

4.2.4 Environments Reconstruction

Figure 4.8 shows the resulting reconstruction of the real town square's loggia described in Section 4.1.2. This reconstruction is the result of a video sequence composed of 200 frames; the frame filter selected 48 frames that are used as input data for our pipeline. In this case we do not have the camera positions ground truth; anyway, we would never be able to obtain a reconstruction with good quality if the SfM result was not accurate enough (see Figure 4.9). In

other words, for the real case, the dense point cloud obtained is a proof of the correctness of the previous phase.

4.2.5 Limitations

As we expected, our pipeline fails when there are not enough feature points that can be tracked in all the views of the window. In the current implementation, there is no recovery procedure that deals with this problem and we suggest some possible solutions in Section 5.1.

(a) Original rectified pair.

(b) Computed disparity map.

Figure 4.6: An example of disparity map computed with our adaptive block-matching algorithm. The black points in the map are points whose disparity is equal to zero or that are occluded in the other image; in both cases, these points are not triangulated.

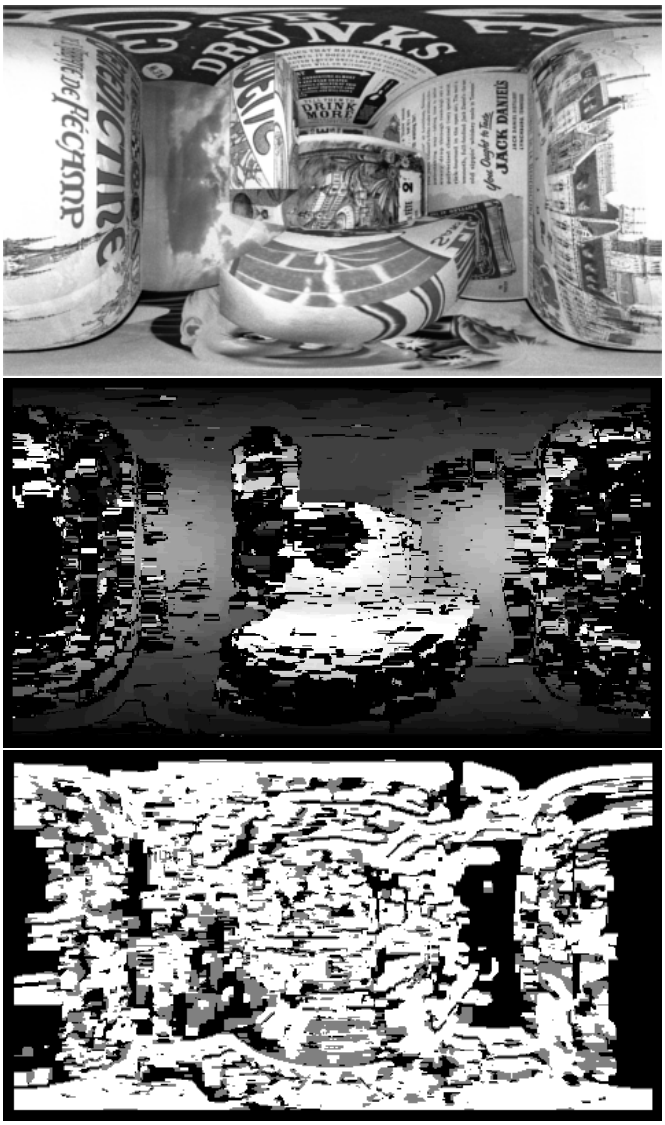


Figure 4.7: Comparison of our adaptive block-matching algorithm's result (middle) with standard block-matching algorithm (bottom) for the same input pair (one of the images of the pair is shown at top). The parameters are the same for both algorithms.

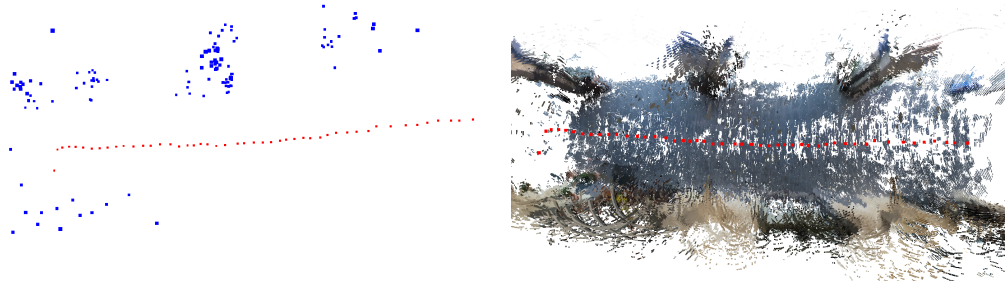


Figure 4.8: The real town square reconstruction: top view of the estimated camera poses as red dots and the sparse points cloud as blue dots (left); top view of the reconstruction for the same environment (with the roof cut out to see the camera trajectory in red) (right). See Fig. 4.9 for the complete reconstruction.

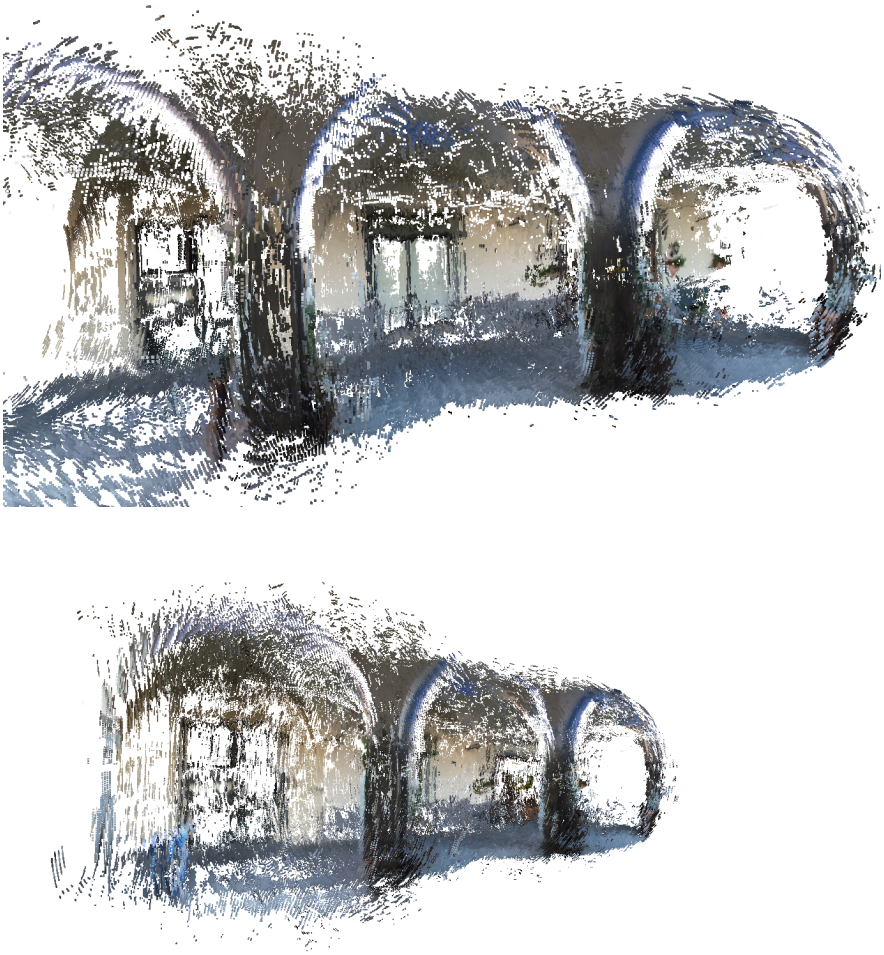


Figure 4.9: Two views of the dense point cloud of the real town square (Section 4.1.2) obtained with our SfM pipeline.

Chapter 5

Conclusion

In this work, we have proposed a SfM pipeline for full spherical cameras that works directly with 360°videos. As we pointed out in Section 1.4 and Section 2.8.3, our contribution includes:

- a simple but effective frame filter that selects the non-redundant frames to be processed based on visual information only;
- a new approach to estimate poses that exploits both frontal and rear points;
- a novel block-matching algorithm for disparity map estimation for equirectangular images.

We have shown the effectiveness of our pipeline by testing it on both computer-generated and real-world environments. Even though our approach is extremely straightforward, especially when compared to more advanced pipelines (e.g., [39], it still offers high-quality results in terms of poses estimation precision and environment reconstruction.

5.1 Future Work

Our SfM pipeline employs existing correspondences between adjacent frames only. This is because we use video sequences as input data, thus the presence of the same features of the current frame in the next one is a natural consequence of

the particular nature of our input. As we pointed out in Section 4.2.5, this particular choice for feature matching can be a problem when not enough feature points can be tracked in a part of the video sequence. In this case, the camera pose estimation is invalid from that point, and it will continue to remain invalid. The current implementation of our pipeline does not deal with this problem, hence, it is up to the user to remove the critical frames and produce two separate reconstructions that can merge together.

Another solution to make the camera pose estimation more robust is to exploit the loop-closure detection as the SLAM algorithms done (Section 2.1). Loop closure is the process of detecting the camera crossing a previously visited place. The identified loop can be used as an additional constraint to improve the camera pose estimation globally.

Another interesting improvement of the pipeline can be to use more robust features detection and matching algorithm more specific for this type of images. Some of the possible alternative algorithms are ASIFT [33], which can perform better in case of views with very tilted cameras, and BRISKS [8], a feature detector inspired by BRISK [21] and designed specifically for full spherical images.

Another approach to improve local motion estimation is to introduce a procedure to extract features that are evenly distributed across an image. The more keypoints are spread over an image, the more accurate is the local motion estimation [16, 39]. The development of this new feature in the current pipeline has just been planned to be finished in short time.

Bibliography

[1] Mohamed Aly and Jean-Yves Bouguet. "Street view goes indoors: Automatic pose estimation from uncalibrated unordered spherical panoramas". In: *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*. IEEE. 2012, pp. 1 8.

[2] Zafer Arican and Pascal Frossard. "Dense disparity estimation from omnidirectional images". In: *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*. IEEE. 2007, pp. 399 404.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *Computer vision ECCV 2006* (2006), pp. 404 417.

[4] Hugh Durrant-Whyte, David Rye, and Eduardo Nebot. "Localization of autonomous guided vehicles". In: *Robotics Research*. Springer, 1996, pp. 613 625.

[5] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381 395.

[6] Mark A Fonstad et al. "Topographic structure from motion: a new development in photogrammetric measurement". In: *Earth Surface Processes and Landforms* 38.4 (2013), pp. 421 430.

[7] Natasha Govender. "Evaluation of feature detection algorithms for structure from motion". In: (2009).

[8] Hao Guan and William AP Smith. "BRISKS: Binary Features for Spherical Images on a Geodesic Grid". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4516 4524.

[9] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Manchester, UK. 1988, pp. 10 5244.

[10] Christopher G Harris and JM Pike. "3D positional integration from image sequences". In: *Image and Vision Computing* 6.2 (1988), pp. 87 90.

[11] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[12] Richard I Hartley and Peter Sturm. "Triangulation". In: *Computer vision and image understanding* 68.2 (1997), pp. 146 157.

[13] Heiko Hirschmuller and Daniel Scharstein. "Evaluation of cost functions for stereo matching". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1 8.

[14] Hugin. *Hugin - Panorama Photo Stitcher*. [Online; accessed 13-October-2017]. 2017. URL: `http://hugin.sourceforge.net`.

[15] Sunghoon Im et al. "All-around depth from small motion with a spherical panoramic camera". In: *European Conference on Computer Vision*. Springer. 2016, pp. 156 172.

[16] Arnold Irschara et al. "From structure-from-motion point clouds to fast location recognition". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 2599 2606.

[17] MR James and Stuart Robson. "Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application". In: *Journal of Geophysical Research: Earth Surface* 117.F3 (2012).

[18] Florian Kangni and Robert Laganiere. "Orientation and pose recovery from spherical panoramas". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1 8.

[19] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 61 70. ISBN: 3-905673-36-3. URL: http://dl.acm.org/citation.cfm?id=1281957.1281965.

[20] Karl Kraus. *Photogrammetry: geometry from images and laser scans*. Walter de Gruyter, 2007.

[21] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. "BRISK: Binary robust invariant scalable keypoints". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2548 2555.

[22] Kenneth Levenberg. "A method for the solution of certain non-linear problems in least squares". In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164 168.

[23] Shigang Li. "Binocular spherical stereo". In: *IEEE Transactions on Intelligent Transportation Systems* 9.4 (2008), pp. 589 600.

[24] Shigang Li. "Real-time spherical stereo". In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 3. IEEE. 2006, pp. 1046 1049.

[25] H Christopher Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections". In: *Nature* 293.5828 (1981), pp. 133 135.

[26] Manolis IA Lourakis and Antonis A Argyros. "SBA: A software package for generic sparse bundle adjustment". In: *ACM Transactions on Mathematical Software (TOMS)* 36.1 (2009), p. 2.

[27] David G Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150 1157.

[28] Chuiwen Ma et al. "3D Reconstruction from Full-view Fisheye Camera". In: *arXiv preprint arXiv:1506.06273* (2015).

[29] Ameesh Makadia, Christopher Geyer, and Kostas Daniilidis. "Correspondence-free structure from motion". In: *International Journal of Computer Vision* 75.3 (2007), pp. 311 327.

[30] Larry Matthies and STEVENA Shafer. "Error modeling in stereo navigation". In: *IEEE Journal on Robotics and Automation* 3.3 (1987), pp. 239 248.

[31] Christoph Rhemann Michael Bleyer and Carsten Rother. "PatchMatch Stereo - Stereo Matching with Slanted Support Windows". In: *Proc. BMVC*. http://dx.doi.org/10.5244/C.25.14. 2011, pp. 14.1 14.11. ISBN: 1-901725-43-X.

[32] Hans P Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover.* Tech. rep. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.

[33] Jean-Michel Morel and Guoshen Yu. "ASIFT: A new framework for fully affine invariant image comparison". In: *SIAM Journal on Imaging Sciences* 2.2 (2009), pp. 438 469.

[34] David Nistér, Oleg Naroditsky, and James Bergen. "Visual odometry". In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on.* Vol. 1. Ieee. 2004, pp. I I.

[35] Alain Pagani and Didier Stricker. "Structure from motion using full spherical panoramic cameras". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on.* IEEE. 2011, pp. 375 382.

[36] D. Scaramuzza and F. Fraundorfer. "Visual Odometry: Part I - The First 30 Years and Fundamentals". In: *IEEE Robotics and Automation Magazine* 18.4 (2011).

[37] D. Scaramuzza and F. Fraundorfer. "Visual Odometry: Part II - Matching, Robustness, and Applications". In: *IEEE Robotics and Automation Magazine* 19.2 (2012).

[38] Adam Schmidt, Marek Kraft, and Andrzej Kasiński. "An evaluation of image feature detectors and descriptors for robot navigation". In: *Computer Vision and Graphics* (2010), pp. 251 259.

[39] Johannes L Schonberger and Jan-Michael Frahm. "Structure-from-motion revisited". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4104 4113.

[40] Roberto Scopigno et al. "3D models for cultural heritage: Beyond plain visualization." In: *Computer* 44.7 (2011), pp. 48 55.

[41] Steven M Seitz et al. "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2006, pp. 519 528.

[42] SPHERON-VR. *SPHERON-VR Website*. [Online; accessed 13-October-2017]. 2017. URL: `https://www.spheron.com/home.html`.

[43] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[44] THETA. *THETA Website*. [Online; accessed 13-October-2017]. 2017. URL: `https://theta360.com`.

[45] Carlo Tomasi and Takeo Kanade. "Shape and motion from image streams under orthography: a factorization method". In: *International Journal of Computer Vision* 9.2 (1992), pp. 137 154.

[46] Bill Triggs et al. "Bundle adjustment a modern synthesis". In: *International workshop on vision algorithms*. Springer. 1999, pp. 298 372.

[47] Ying-mei Wei et al. "Applications of structure from motion: a survey". In: *Journal of Zhejiang University SCIENCE C* 14.7 (July 2013), pp. 486 494.

[48] MJ Westoby et al. "'Structure-from-Motion'photogrammetry: A low-cost, effective tool for geoscience applications". In: *Geomorphology* 179 (2012), pp. 300 314.