POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Master of Science in Computer Science and Engineering

Immersive mobile augmented reality application for mountain peak recognition

Supervisor: Prof. Piero Fraternali

Master Graduation Thesis by:
Antonio La Salandra
id. 842219

Academic Year 2016/2017

Abstract

Thanks to the recent great evolution in the mobile sector most people may now use devices with great computing power and hardware equipment and perform tasks that would have been possible only on computers and dedicated hardware just a few years ago. Nowadays, by using a smartphone, it is easily possible not only to acquire a large number of information but also to process and execute heavy multimedia contents. For example, with the help of sensors available on a smartphone, it is possible to quickly and easily capture video, audio, geographical information such as latitude, longitude with high quality and resolution. Furthermore, thanks to the increasingly sophisticated modern CPU and the growing memory availability, it is possible to quickly combine these information and produce multimedia contents that can be used in innovative ways, for example in augmented reality or virtual reality environments.

Leveraging the technological capabilities of new smartphones, their widespread diffusion and massive use, this thesis deals with the development of an Android application that enables: to take panoramic photos of mountain landscapes with additional information on geographic position, orientation and field of view, retrieving data from sensors; the automatic and high-precision peak identification in the photos; the innovative viewing of panoramic images, annotated with the detected peak information, in an immersive environment using the Google Cardboard virtual reality viewer; the sharing of the contents on social networks, with particular attention to those that enable the viewing of panoramic photos in an immersive way.

By providing the user with a powerful tool that can generate multimedia contents in a simple, accurate and automatic way, this work can have many social applications. The great amount of content that can

be produced may be used for monitoring environmental resources and building collective intelligence systems, as well as for automated mapping of natural sites and the generation of virtual tours in not easily accessible places.

Sommario

L'evoluzione del settore mobile degli ultimi anni ha permesso di avere sempre a portata di mano dispositivi con grande potenza di calcolo e una ricca dotazione hardware che permettono operazioni che solo qualche anno fa sarebbero state impossibili, se non con computer ed hardware dedicato. Al giorno d'oggi utilizzando uno smartphone è possibile l'acquisizione di un gran numero di informazioni e l'elaborazione e fruizione di contenuti multimediali pesanti. Ad esempio grazie ai sensori presenti ormai su ogni smartphone è possibile acquisire in modo semplice e veloce informazioni video, audio, informazioni geografiche come latitudine e longitudine con una qualità e risoluzione quasi professionale. Inoltre, grazie alle moderne CPU sempre più performanti e alla crescente disponibilità di memoria, è possibile combinare velocemente queste informazioni e produrre contenuti multimediali fruibili in modi innovativi, come ad esempio in ambienti di realtà aumentata o di realtà virtuale.

Sfruttando le possibilità tecnologiche messe a disposizione dai nuovi smartphone, la diffusione capillare e l'utilizzo massiccio che se ne fa, questa tesi documenta lo sviluppo di un'applicazione Android che permetta: l'acquisizione di foto panoramiche in zone montuose, geolocalizzate e corredate di informazioni aggiuntive ricavate tramite sensori, come angolo di campo e orientamento; l'identificazione automatica e ad alta precisione dei picchi presenti nelle foto; la visualizzazione delle immagini panoramiche, annotate con le informazioni sui picchi rilevati, in modo innovativo in un ambiente immersivo utilizzando il visore per la realtà virtuale Google Cardboard; la condivisione dei contenuti sui social network, con particolare attenzione a quelli che permettono la visualizzazione di foto panoramiche in modo immersivo.

Fornendo all'utente finale un potente strumento in grado di generare

contenuti multimediali in modo semplice, preciso ed automatico questo lavoro può avere diverse applicazioni in ambito sociale. La grande quantità di contenuti prodotti potrebbe infatti essere utilizzata per il monitoraggio delle risorse ambientali e la costruzione di sistemi di intelligenza collettiva nonché per la mappatura automatica di luoghi naturali e la generazione di tour virtuali in luoghi non facilmente raggiungibili.

Acknowledgements

First of all I would like to thank Prof. Piero Fraternali for giving me the opportunity of working on such an interesting and exciting project and for his constant support during the preparation of this thesis.

I would also like to thank all my friends and colleagues who have helped and contributed to this work.

The biggest thanks are for my family. My parents, Giuseppe and Ida, have always supported my choices and have encouraged me during all these years: they have been the best guide I could ever have had. My Grandmother Laura and Uncle Gerry have followed all my school experiences with utmost care from the beginning up to the end. At last my special thanks are for my love, Chiara: she has always been by my side during these years, sharing my successes as well as my failures, proving to be the best mate I could ever have met for my journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes the development of an Android application that enables: to take panoramic photos of mountain landscapes with additional information on geographic position, orientation and field of view, retrieving data from sensors; the automatic and high-precision peak identification in the photos; the innovative viewing of panoramic images, annotated with the detected peak information, in an immersive environment using the Google Cardboard virtual reality viewer; the sharing of the contents on social networks, with particular attention to those that allow the viewing of panoramic photos in an immersive way.

In this work some applications that the technologies developed in the context of this project may have on the social level will be discussed, such as: monitoring environmental resources and building collective intelligence systems or automatically mapping natural sites and creating virtual tours in not easily accessible places.

The input for this work as well as an important technological and innovative contribution have been given by a research project started by the Politecnico di Milano few years ago and still under development: SnowWatch. In this chapter we will take an overview of the project, the scientific and technological innovations introduced and how and why, starting from SnowWatch, the idea of this work was born.

The areas touched by this project are many: image processing applied to the field of panoramic photography, computer graphics applied in the virtual reality field, the study of the Android platform and of the

design guidelines for an application able to reach as many devices as possible, the study of theories on human computation to stimulate the users' engagement and their participation. In this chapter the various sectors involved will be introduced and discussed in a broader sense to give an idea of the context in which the work is centred. More details will be given in the next chapters.

1.1 Background: The SnowWatch and PeakLens projects

SnowWatch is an environmental monitoring project that has been funded by the European Union and carried out by the Web, Data and Society research group of the Politecnico di Milano [20]. The purpose of the project is the monitoring of the availability of water resources present in the mountains in the form of snow using user generated photos taken from social networks as input in place of or in addition to satellite photos and photos taken from fixed cameras placed in several mountain areas. This is done through a deep analysis of the content of the photos that makes the extraction of information of interest possible. In this project an algorithm capable of extracting the skyline from a mountain landscape photo, distinguishing the skyline from obstructing objects or clouds, has been developed. Thanks to this algorithm it has been possible to create a system that after the extraction of the skyline representation from a geolocated photo is capable to compare and align it with a tridimensional reconstruction of the visible landscape from the point in which the photo has been taken and to estimate some parameters such as the camera orientation and field of view. The tridimensional model of the Earth surface is built by using the DEM (Digital Elevation Model), that is a collection of latitude, longitude and altitude values detected by satellites in a grid of points uniformly spaced on the Earth's surface. Once the data on orientation and field of view have been estimated, the system is able to determine visible peaks positions and insert tags with the names of the peaks in the detected positions.

Later it has been decided to give users an active role in collecting photos and therefore to develop a mobile application that, by using the augmentid reality as a means of displaying information, provides the

use of the peak identification service shown above in real time. From this idea the application for Android devices, named PeakLens, was born [15]. PeakLens enables peak identification in real time and with great precision, using as input the frames captured by the smartphone camera, user position and camera orientation obtained through smartphone's gps, gyroscope, magnetometer and accelerometer. A complete description of the technologies developed and how these have been applied to the field of the augmented reality can be found in [29]. By providing users with a pleasant and immersive experience through the use of augmented reality, the goal is to stimulate the production of a content suitable to environmental monitoring systems. Photographs generated through PeakLens are in fact perfect for this kind of systems because they have metadata on location, orientation and field of view obtained through sensors. In turn sensors are subsequently improved through the skyline extraction and alignment algorithms, which overcome sensor errors and inaccuracies.

Given the technological assumptions and the ongoing process of active involvement of users in collecting the material required for the environmental monitoring system previously described, the idea of this work was born. The application (whose development is documented in this thesis) exploits the current trend of virtual reality utilization for 360 degree images and videos to create another user-generated content channel. Thanks to the engagement created by this kind of immersive environment, the user is encouraged to use the application and create a useful content for collective intelligence and environmental monitoring systems. The kind of content generable by this application is better than PeakLens photos for these systems. By the application it will be possible to create panoramic photos with all the metadata and information already present in PeakLens, which will capture a very wide area or even all the surrounding environment. Another possible application of the photo collection generated by this application could be the automatic generation of virtual tours in mountain areas in Google Street View style. Users can then have access to these tours through a web application and browse the material, both in a classic and immersive way, using a virtual reality viewer, moving from one panorama to another by clicking on hotspots, displaying peak information. Users can also see the changing of the landscape over time, when more photos taken at the same point but at different times are available.

1.2 Panoramic Photography

Panoramic photography is a photographic technique that allows the generation of images with a wide visual angle, using specialized equipment or software [14]. Panoramic pictures provide users with a view of an environment closer to reality. The most common method to produce a panoramic photo, using digital photography, is to take a series of photos with a degree of overlapping (between 30% and 50%) and then combine them to create the panorama. To take a photo you can use specific cameras such as the Ricoh Theta S [19] (which offers specific high quality hardware and a built-in auto-composing software) as well as common cameras used with a levelled and graduated tripod to adjust the camera position during the shots and composing software like Hugin [11] to create the ultimate panorama or even a smartphone. Thanks to the increasing power and the particular attention for the photographic components on modern mobile devices in the last years, it is now possible to take good quality panoramic photos using a simple smartphon or tablet.

The process of building the panorama is called stitching in technical jargon and consists of two main processes, divided in turn into several optional sub-processes: photo alignment and stitching. Alignment, which always precedes stitching, consists in identifying the relative positions of the photos in the sequence. These can be defined manually as well as identified by software. Stitching is the phase in which the final view is generated as a graphic composition of the individual initial images, using the data calculated in the alignment phase. After the stitching phase, it should not be noticeable that the final view was generated by combining several images. There are several factors to be considered about the generation of good quality panoramic images: the movement of the camera during shooting (which may be the source of alignment and parallax errors), the presence of moving objects in the scene, color variations and exposure among the different pictures. All these factors require the application of several processing steps to generate a semi-professional panorama. The process of generating panoramic images is generally very expensive in terms of resources and has a high complexity both in space and time. This is because of very heavy input data (photos with increasingly higher resolutions) and the different steps required for processing this type of data.

In this work a fully integrated stitching module has been developed, both considering the limited resources of a mobile device compared to a traditional computer and the availability of additional information compared to a classic digital camera, such as the directional orientation data provided by gyroscope and accelerometer. Particular attention has been given to the interface and user's guide to minimize errors due to incorrect movements during the acquisition phase. In this work panoramic photos are the starting point for giving the user an immersive experience using virtual reality. The photos generated by our application contain metadata on the user's location and about camera orientation and field of view. These data and the photos will be the input for the automatic peak identification module that will enable the peaks detection in landscapes.

In the next chapter the state of the art of panoramic photography will be discussed in detail and a major focus will be given to the case of mobile devices, the applications and services currently available and to the reason for an implementation of our own stitching module inside the final application.

1.3 Virtual Reality

The term virtual reality is used to describe a three-dimensional computer generated environment that a human being can perceive and interact with in a way similar to what happens in the real environment.

Our perception of the world rises from the information generated by our senses and the way we combine and process this information in our brain. Thanks to our senses the brain receives a great deal of high frequency and high precision information and, by combining this information, it generates our perception of reality. Depending on the type of information received and the acquired knowledge of reality, our brain will generate an answer. This is the mechanism at the basis of the perception and interaction with the surrounding environment. The goal of virtual reality is to somewhat simulate this kind of experience in the context of a digital, computer generated environment [24].

Virtual reality systems use some hardware devices to "place" the user

in the three-dimensional environment. The main device is the virtual reality viewer. This may be a helmet or some eyeglasses with some screens inside, close to the eyes, on which to project the digital world. It is very important that the viewer isolates the user from the outside environment and that the screens and lenses make it possible to have a wide visual field as close to the human one as possible. These viewers are equipped with a gyroscope to detect the user's head movements and to make movements within the three-dimensional environment match to physical ones. Besides the viewer users can also use headphones to play 3D spatial sound as well as controllers or gloves with a tactile feedback to interact with the virtual environment [18]. All these solutions are designed to deceive our brain and to make the virtual experience as immersive as possible. The biggest problems in the virtual reality context are due to the frequency, latency and resolution of the information presented to the user. For example, our view is able to provide the brain images with quality and resolution far superior to what is possible with screens used in viewers for virtual reality. The latency introduced during the processing of information as a result of an interaction might be far superior to what happens in reality, thus involving "wrong" feedback to the user. In recent years virtual reality research has geared towards improving hardware and software performance in order to bridge the gap between real and virtualized experience.

In our application we have decided to take advantage of the growing trend in recent years in the use of virtual reality, especially with regard to viewing 360-degree images and videos in order to provide the user with an innovative way of viewing the multimedia content generated through the application. In this way the user can view the panoramic photos in an immersive way and interact with the surrounding environment by clicking on the markers automatically placed on the peaks identified in the image and displaying the information available about the mountain he is watching.

1.4 Human Computation

Human computation can be defined as:

> *" ... a paradigm for utilizing human processing power to solve problems that computers cannot yet solve. "* Luis von

Ahn [45]

The main goal of this work is the creation of another channel for the acquisition of data useful for environment monitoring purposes. Users are provided with a pleasant user experience to stimulate the use of the application and the production a useful content for an environment monitoring system. The most suitable paradigms for representing this kind of approach are: crowdsourcing, social computing and collective intelligence. These paradigms are correlated one to each other under certain aspects, referring in some way to the field of human computation. A possible taxonomy of human computation, displayed in Figure 1.1, has been produced by combining definitions given by [40] and [32]:

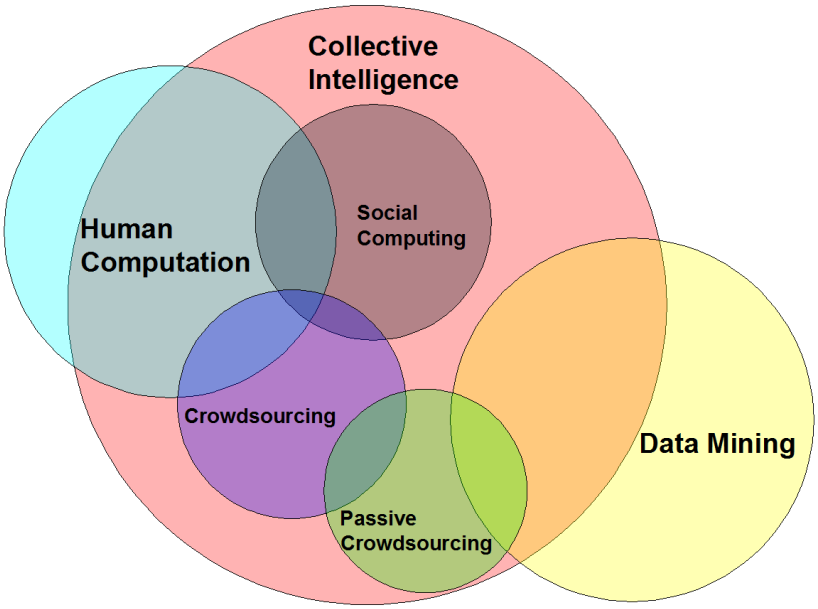


Figure 1.1: Proposed taxonomy of human computation including all related paradigms.

- *Crowdsourcing*: This concept, deriving from outsourcing, is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people. People can usually work for free as volunteers (es. wikipedia) or in exchange for a small fee. Our system is based on this concept since people get involved into using our application and their effort is indirectly exploited for further environmental analysis.

- *Social Computing*: In a system based on social computing people are facilitated to share and exchange multimedia information by means of information technology systems. The scopes can be broad and various but in general they do not involve computation but social interactions. In particular our system is a social computing example helping people identify peaks and stimulating social interaction, too. Examples can be the sharing of multimedia information, as the photographs, and a contribution to the environment safeguard.

- *Collective Intelligence*: Collective intelligence consists in platforms where groups of people collaborate in an "intelligent" manner altogether with a common general purpose. These platforms can be seen as a network to connect citizens and ideas for social innovation, leveraging on collective intelligence and actions to address sustainability challenges. This paradigm could be used to build systems where users will have an engaging and socially rewarding experience and at the same time generating a stream of data useful for environmental monitoring purposes.

- *Game With A Purpose*: This line of work focuses on exploiting the billions of hours that people spend on line playing with computer games to solve complex problems that involve human intelligence [36]. The emphasis is on embedding a problem solving task into an enjoyable user experience, which can be conducted by individual users or by groups [32]. Gamification is a crucial part of our project that will be taken into account more closely further on and some ideas of it will be presented in Chapter 6 about Future Enhancement.

- *Human Sensors*: Exploiting the fact that mobile devices tend to incorporate more and more sensors, this approach deals with a real-time collection of data (of various natures) treating persons with mobile devices as sensors for the data. Examples of these applications are earthquake and other natural disaster monitoring, traffic condition control and pollution monitoring.

1.5 Document Structure

In the next chapter the state of the art of the main areas involved in this work will be discussed and some relevant applications for each area of interest will be analyzed. In Chapter Three the whole application is presented and the solutions adopted to build the different parts of the system are described and analyzed in detail. In Chapter Four the implementation architecture is described and then the implementation of each module is explained. In the fifth chapter the experimental study that has been done to evaluate the precision of application results is described. Finally in the last chapter the final conclusions are exposed together with some considerations on the possible directions for the future improvements of this work.

Chapter 2

Related Work and State of the Art

The main areas involved in this work are panoramic photography, virtual reality and software aided environmental monitoring. In this chapter the main technological solutions currently available in these areas will be discussed with a special focus on mobile devices. Some applications currently available on the market and their role in the current scenario will be defined.

2.1 Panoramic Photography

Panoramic photos are the starting point for the virtual environment built by the application and the main input for the peaks identification module. This module is based on the comparison between the skyline extracted from the panorama and the skyline generated using the digital elevation model of the terrain. Artifacts and distortions introduced during the stitching process might alter the final results and cause inaccuracies in the positioning of the peaks. It is very important therefore to be able to compose good quality panoramic photos that do not show significant distortions and alterations.

2.1.1 Image Alignment and Stitching techniques

Image alignment and stitching are amply faced problems in the image processing and computer vision fields. [43] presents an overview of the various techniques and studies done in these fields over the years.

The process to compose a panorama from a number of pictures with a certain degree of overlap is divided into two main phases: the alignment or registration phase and the stitching phase.

Image Alignment

The goal of this phase is to discover the correspondence relationship among the images of the sequence. The first step is the choice of the appropriate mathematical model relating pixel coordinates in one image to pixel coordinates in another one. Images can be correlated by different kinds of transformation: from 2D planar transformations to 3D transformations in the space. Figure 2.1 shows an example of possible 2D planar transformations.

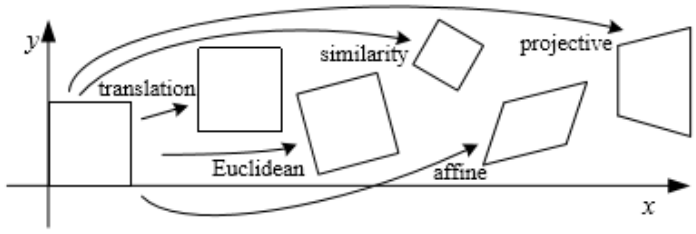


Figure 2.1: 2D planar transformations.

After the definition of the motion model the next step is to estimate the parameters that describe the movement between each pair of images of the sequence. There are two main techniques to achieve this goal: the direct or pixel based alignment and the feature-based registration.

The *pixel-based alignment* consists in a direct comparison of the intensities of the image pixels. This is performed by shifting or warping the images relative to each other and looking at how much pixels agree. In order to apply this kind of method an error metrics and a search technique must be chosen to respectively evaluate the quality of the alignment and define how to search for a possible solution. The most simple technique might be to try all the possible positions, but in practice this may be too slow to be applicable. In [43] some possible error

metrics, search techniques and their pros and cons are depicted.

The *feature-based image registration* consists in the extraction of some distinctive features from images, the matching of the extracted features to identify a global correspondence between them and the estimation of the geometric transformation between images, Figure 2.2. A feature is an image pattern which differs from its immediate neighborhood and is generally associated with a change of an image propriety (e.g. intensity, colour, texture) [44]. Given two images of the same scene captured from different points of view the first step is the detection and description of a number of such features. The description phase consists in the definition of a representation that univocally identifies each detected feature. Features that correspond to the same region in different images and from different perspectives should be recognizable. In [37] an algorithm for the detection and description of features invariant to scale, rotation and illumination is presented. After the definition of the two sets of features the next step is to determine which features come from corresponding locations in different images and the motion parameters for the transformation between images. For this purpose the RANSAC algorithm [31] is usually used.

Figure 2.2: An example of the feature-based image registration process.

The feature-based registration may lead to errors when a good amount of features is not identifiable, e.g. images not textured enough or with unevenly distributed features. Pixel-based methods use the available information optimally since they employ the contribution of every pixel

in images. The biggest disadvantage of these methods is that it is almost impossible to try and evaluate all the possible transformations between the two images. A good solution might be first to estimate the transformation by using a feature based approach, then warp the images to a common reference surface and refine the alignment using a pixel-based method. In this work a similar approach has been used. This will be discussed in the following chapters.

Image Stitching

In this phase the aligned photos are merged to compose the final panorama. Several techniques are applicable in the various steps of the stitching process in order to reduce artifacts due to misalignment errors, moving objects as well as color and exposure differences between pictures. An example of the stitching process is provided in Figure 2.3 The main steps are:

- *Color correction* and *exposure compensation:* Photos are usually taken with automated settings, which produces differences in exposure and white balance as the illumination changes across the scene. In this phase some techniques are used to correct color differences between images. In [48] and [33] two possible solutions that can be implemented on mobile phones are presented.

- *Optimal seam estimation:* The aim of the algorithms used in this step is to find the best path along which to cut and then merge the two images. This is useful to avoid in the next phase ghosting problems due to moving objects and spatial alignment errors. The two main applicable algorithms are based on graph cut or dynamic programming. [8] provides a comparison between the two different approaches.

- *Blending:* It is the phase in which images are merged. After this step it should not be possible to notice that the image is composed of several images. There have been many studies on this subject and a lot of techniques have been produced. These techniques differ in computational load and quality of the product result. They range from very fast and simple solutions to solutions with high computational complexity. In [12] the main techniques applicable in this phase are discussed.

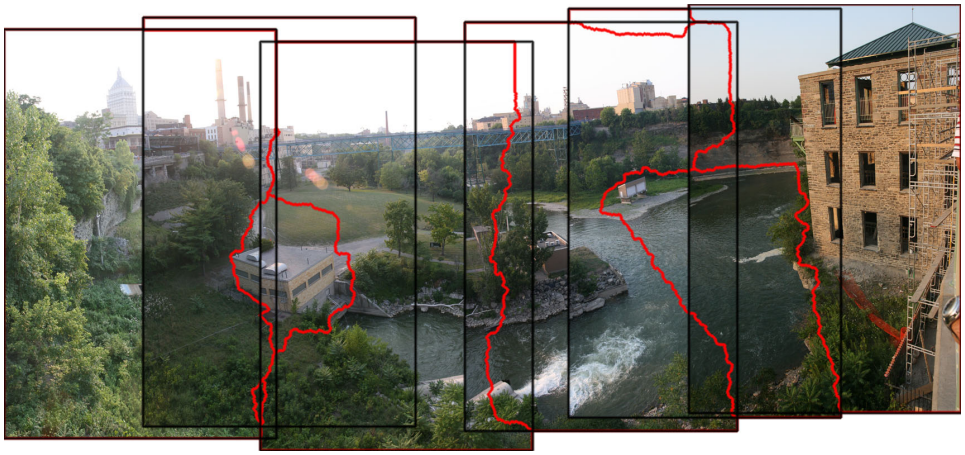


Figure 2.3: An example of the image stitching process by using the seam-estimation technique.

In [25] a complete pipeline that fully automates the stitching process is presented. This approach is quite good because it is able to automatically recognize overlapping images and to compose panoramas in unordered datasets. It provides high-quality results by using gain compensation to reduce color differences and multi-band blending to seamlessly blend the images. This is a great solution for computers but it may cause some problems in a mobile device implementation. The main drawbacks are the high computational requirements and the large amount of necessary memory. Algorithms need to keep all the images in memory and this might lead to a quick run out of memory on a mobile device. In a mobile context lightweight algorithms to reduce memory requirements and to boost performances are needed. In [34] a complete pipeline for mobile devices that includes both the alignment and stitching phase is presented. A solution for the stitching phase that can produce very high-quality panoramas is proposed in [48]. In the next two chapters the techniques used in our application and their implementation will be described in detail.

2.1.2 Types of panoramic photos

Different kinds of panorama can be generated depending on the chosen surface on which to compose it. There are three main typologies: planar, cylindrical, spherical or equirectangular [22].

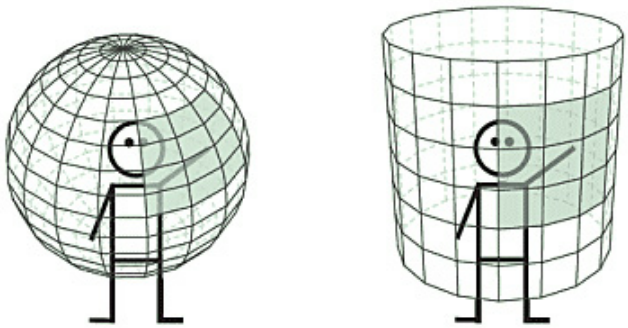


Figure 2.4: Cylindrical and spherical panoramas.

Planar Panorama

Planar panoramas are generated by a translation of the camera along the scene until the whole subject is acquired. This is the least common type and cannot be used for the construction of an immersive environment. So it has not been taken into account in this work.

Cylindrical Panorama

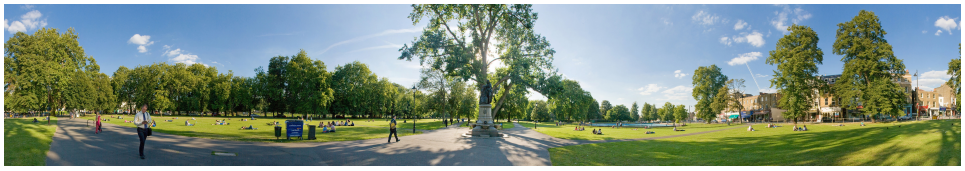


Figure 2.5: Cylindrical panorama.

Cylindrical panoramas are generated by a pure rotation of the camera around the vertical axis. Photos are projected and composed inside a cylindrical surface. This kind of panorama can be used in the context of virtual reality applications and it provides users with a complete view of the whole surrounding environment except for the top and bottom. This is the kind of panorama used in this work because of the easy acquisition when using a smartphone (pure horizontal rotation) and because it contains all the useful information for our system (mountain skyline).

Spherical or Equirectangular Panorama

Spherical panoramas are generated by a rotation of the camera around

Figure 2.6: Spherical or equirectangular panorama.

all the three axis in order to acquire all the surrounding environment including the top and the bottom. These kinds of panoramas are often used in virtual reality applications because they provide a complete view of the surrounding environment. These are easily acquirable with proper cameras but the process is slower and more complex when using a smartphone. An interesting kind of spherical panorama is the equirectangular stereo panorama. This is made up of two spherical panoramas of the same scene with small variations stacked. These are used one for the left and one for the right eye in the rendering process to produce a stereoscopic effect.

2.1.3 Panoramic Photography applications

Nowadays there are a lot of solutions for generating panoramic photos. It is possible to purchase specific 360-degree cameras or use conventional cameras and desktop software for stitching. Our study, however, focuses on the solutions available for mobile devices. The main goal is to reach the widest possible number of users in order to produce a relevant data stream as input for an environmental monitoring system. Smartphones are the best channel for this scope.

The panorama feature has been implemented in a lot of stock camera apps. The most interesting examples can be found directly in

the Android stock camera application. This application provides two panorama modes: panorama and photosphere. The panorama mode enables the acquisition of horizontal, vertical, planar or even semi-spherical panoramas. It provides a very simple interface that guides the user during the shots and is capable of generating high-quality panoramas. The photosphere mode, which is also embedded inside the Google Street View application [7], enables one to shoot complete 360 spherical panoramas and share them on Google Maps. The process is a bit tricky in comparison to the panorama mode because of the major complexity of the movement to create the full picture. On the Street View website [4] Google provides information and tips on apps and equipment that can be used to produce panoramic photos sharable on Google Maps. Another interesting application is Cardboard Camera [5], also produced by Google. With this application it is possible to shoot cylindrical stereoscopic panoramas and view them in an immersive environment by using the Google Cardboard VR viewer. All these applications can generate good quality panoramas but they are not always usable in the context of this work. Undoubtedly users might upload panoramas generated with other applications in PeakLensVR to exploit the peaks identification and 3D viewing functionalities. But this possibility has been discarded in the present work for two reasons. The first is that, to work properly, the peaks identification module needs some additional information, such as: latitude and longitude, camera orientation, horizontal and vertical field of view. Although panoramas generated with some apps (e.g. Photosphere in Street View) are equipped with some of these metadata, there is no guarantee about their complete presence and their format. The second reason is that we wanted to build a complete application to provide users with all the needed functionalities, from the panorama acquisition and composition to the 3D viewing of pictures. In this way users will not be bound to other applications for the contents production and may also have a pleasant, complete user experience.

In this work a module for the panorama acquisition and a nice interface to guide users during the shot have been implemented. In the next three chapters this solution will be shown in detail and results will be evaluated. In the final chapter some possible future improvements will be discussed.

2.2 Virtual Reality and Immersive Environments

The trend of recent years has been the use of smartphones as a main component of the VR viewers. Modern VR viewers are in fact just supports in which to place the smartphone. All the work is made by the smartphone itself. This has been made possible by the increasing computing power of these devices and the quality of their hardware components. The main factors to be considered for producing a good device for virtual reality are:

- *High-density* and *large size displays*, on which to project the 3D world in order to render a realistic and non pixelated environment.
- *High-frame rate* and *low-latency*, in order to render the scene fluidly and allow a smooth interaction.
- *High-precision*, in order to identify and reproduce user movements in a precise and reliable way.

Modern smartphones with very high-performance CPU, full-hd or even 4k displays as well as gyroscope and accelerometer meet these requirements perfectly. By using smartphones instead of buying expensive hardware virtual reality has become everyone's reach. This has been the driving force at the basis of the growth and diffusion of this field in recent years.

2.2.1 VR devices: typologies and the choice of Google Cardboard

VR platforms are divided into two categories: the ones that work by using a pc as computational unit and the ones that work using a smartphone. The first category usually has better hardware than the second and high computational power exploiting computers and high-end graphic cards. On the other side the second category is more affordable and usually good enough for occasional and non professional use. Some platforms of the first category are: the Oculus Rift and the Htc Vive. Even if these platforms provide better quality, the focus of this work has been on the second category. The reason is that PeakLensVR is going to to be an all-in-one application to acquire and view contents. So the only choice was to use a platform based on smartphones. The two main choices in this category were: the Gear VR and Google Cardboard. The

Figure 2.7: VR headsets: Oculus Rift, Htc Vive, Samsung Gear VR, Google Cardboard.

Gear VR is produced by Samsung in collaboration with Oculus and it works with Samsung's flagship smartphones. The Google Cardboard platform provides developers with some SDKs to develop an immersive application both on Android and iOS. Unlike other companies, Google does not produce its own headset, but has set up specifications to build it [6]. One can build its own headset by using cardboard, some lenses and felt or buying a premade one. While the Gear VR can be used only with few smartphones, those built according to Google specifications can be used almost with every smartphone. Moreover, while the price of the Gear VR is about 100€, there are several companies that produce cardboard headsets with prices ranging from 5€to about 100€. These differences depend on the materials with which they are constructed and consequently their visual quality. Actually there is another platform, and another headset, for VR as an alternative: the Daydream. Daydream is the new platform for high-quality virtual reality created by Google and released in recent months. The Daydream headset works with the latest smartphones produced by Google: Pixel and Pixel XL. This platform was not yet available when this work started, so it was not taken into account. With the exception of Cardboard all previously mentioned headsets are equipped with a dedicated

controller. As in the case of headsets there are several controllers produced by different companies and usable with these ones. In addition, some headsets have a lateral button in order to perform very simple interactions within the digital environment.

Although the Gear VR quality is slightly higher than that of the Google Cardboard, the platform chosen for this work was the latter one. Thanks to the support of almost every smartphone and the wide range of prices this is the most suitable platform for reaching the largest number of users.

2.2.2 VR and Immersive Environments applications

In recent years there has been a great production of applications and technologies for virtual reality. Nowadays it is possible to view photos, videos, to play games in an immersive environment almost on every platform: websites, mobile and desktop applications. In addition to gaming the growth direction has been to make immersive multimedia contents, such as photos and 360 degree videos, easily accessible to users.

A lot of applications, already present on the market, have given the possibility of viewing their content with VR headsets. By using the Street View app it is possible to explore places and to see available photos in VR mode. The Youtube application offers the possibility to see all the videos in an immersive environment. Facebook, too, has recently launched its own platform for 360 degree photos and videos [1]. Now it is possible to share on the social network photos and videos, made with 360 cameras or even with smartphone applications that use the Photo Sphere XMP metadata [16],[2], and view them using the smartphone as a viewport on the digital world. It is possible to see contents in fullscreen mode and look around by moving the phone (using sensors to capture movements) or by simply clicking and dragging. Facebook uses data such as the field of view and orientation to build the scene in a way similar to what PeakLensVR does. PeakLensVR uses these metadata to enable one to share panoramas generated on Facebook. Maybe in the future the possibility of importing pictures generated with applications that use Photo Sphere XMP metadata will be implemented in our application, too. An application to be mentioned in this

context is Cardboard Camera. This differs from the others because it generates and allows users to see stereoscopic panoramas. Thanks to the realistic effect created by these panoramas, the user feels more involved in the virtual experience. This could be an interesting feature to be implemented in PeakLensVR future updates. Given the increasing popularity of 360 degree photos and videos, Google has built some tools to easily embed these contents on mobile applications and websites. It made a component named "VR View" that can be implemented both on Android and iOS applications by using the Google VR SDK. This gives the possibility of seeing media in fullscreen immersive and VR modes almost without any coding. All the logic to build the 3D scene is inside the VR View and transparent to the programmer. The immersive mode, Figure 2.8 provides the same experience of Facebook 360 while the VR mode 2.9 enables one to view media by using a Cardboard headset.

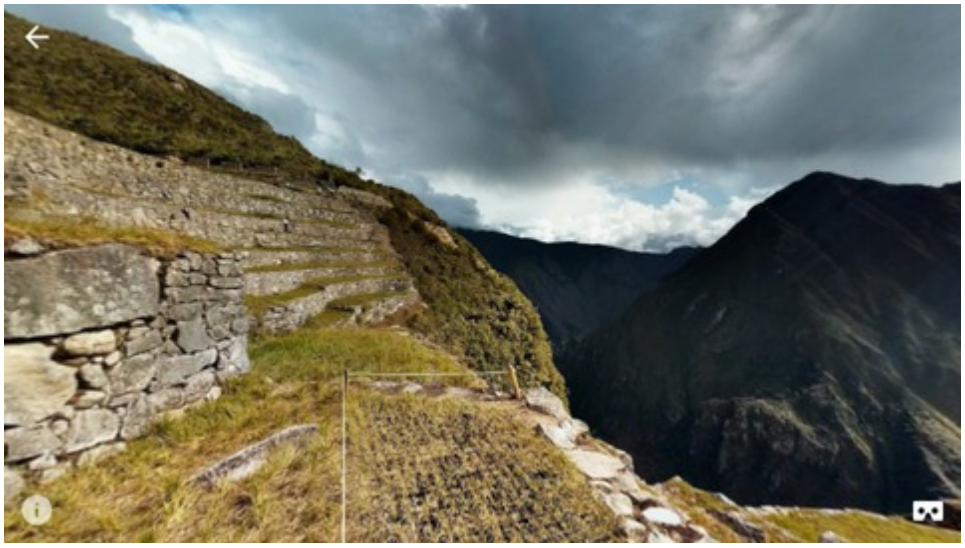


Figure 2.8: VR View fullscreen immersive mode.

VR View can be also implemented on the Web by using a JavaScript API that creates and controls the contents of an iframe. Google is constantly working on these tools and it has recently introduced the possibility to create hotspots users can interact with. Now this is possible only for the Web version of the VR View. In PeakLensVR users are provided with the same kind of experience. It is possible to view panoramas in fullscreen immersive mode or in VR mode. In addition it

Figure 2.9: VR View Cardboard mode.

is possible to click on hotspots placed in peaks positions and to see 3D dialogs with all the available information about the peaks . An interesting project in the field of immersive experiences, which is somehow related to this work, is Trail Me Up [21]. This is a Web platform made by an Italian startup that offers virtual tours in Google Street View style in places accessible only by walking. They made a device for the automatic mapping of trails, a backpack with a camera mounted on it that automatically shoots geolocated photos while walking. Then these photos are processed to create the virtual tour. Sometimes in these tours hotspots are manually placed on points of interest such as mountains. In future it may be possible to automatically build virtual tours such as the ones built by Trail Me Up by using data generated by PeakLensVR users. Furthermore hotspots in points of interests (such as mountain peaks) may be placed automatically through the image alignment and peak recognition technology, developed during the SnowWatch project.

2.3 Environment monitoring mobile applications

The final scope of this work is the introduction of a new channel for the production of data suitable for environmental monitoring purposes. The value of user-generated data for this scope has grown with the pos-

sibilities made available by the technological evolution over the years. The involvement of people for environmental monitoring purposes began with the birth of *citizen science*. In [28] an exhaustive explanation regarding the evolution of this field over the years and the involved technological factors is provided. Citizen science can be described as "a process where concerned citizens, government agencies, industry, academia, community groups, and local institutions collaborate to monitor, track and respond to issues of common community [environmental] concern" [46]. The Earthwatch Institute, founded in 1971, is one of the oldest documented citizen science entities. The main goal was to offer volunteers the opportunity to join research teams through the collection of field data in some environmental and ecological monitoring areas. Nowadays, after the advent of the social web and mobile devices, the citizen science has taken several steps forwards. The volume and the availability of user-generated data have grown exponentially thanks to the sharing on social platforms and the massive use of smartphones as multimedia content producers. In addition, even the value of these data has grown. The presence of more and more sensors on smartphones has given rise to the production of data containing different kinds of information (e.g. photos with metadata on position, orientation, timestamp etc.), making the process of extracting knowledge more accurate. As consequence there has been the release of several mobile applications for environmental monitoring [30]. Some examples are listed below. One is project Budburst. It aims at gathering information regarding the flowering of native plants to study the climate change [41]: mobile phones are used by volunteers to upload timestamped, geotagged plant photographs. eBird is a project whose aim is to harness the power of everyday birders in an effort to better understand bird distribution and abundance across large spatio-temporal scales and to identify the factors that influence bird distribution patterns [42]. This is achieved by providing birders with some tools (web platform and mobile apps) to record and share their observations with a large community of citizens and scientists. CrowdHydrology is another project whose aim is to collect spatially and temporally distributed hydrologic data at an inexpensive cost by using measures voluntarily sent by citizens through text messages to a server that stores and displays the data on the web [38]. NoiseTube is a project based on the idea to transform smartphones into noise pollution detection sensors to help

citizens to measure their personal exposure to noise in their everyday environment [39]. In [35] the same mobile sensing approach is applied to meteorology monitoring.

PeakLensVR fits perfectly in this context. Panoramic images produced through the application can be used to observe very wide areas and to identify and model some environmental factors, e.g. the presence of water analyzed in SnowWatch. By means of positional metadata and timestamps it is possible to spatially and temporally index images and extract some additional useful information as that about the peaks positions in the image. That is the case analyzed in this work.

Chapter 3

Problem Statement and Proposed Approach

The goal of this work is to build a mobile application that:

- allows users to shoot panoramic photos of the surrounding environment equipped with metadata on geographic position, orientation and field of view;
- uses these data and some image processing algorithms to automatically identify the mountain peaks in the images;
- allows users to visualize panoramas in an immersive environment by using the Google Cardboard virtual reality viewer and to share them on social networks with particular attention to those that support the immersive visualization of panoramas.

In this chapter the adopted approach to build the whole PeakLensVR application is presented. All the phases will be described in detail, from the acquisition of the surrounding scene to the VR environment construction and visualization.

3.1 Overview of the application

The application consists in four main modules that work independently of each other. Each one corresponds to a specific phase of the application workflow. In Figure 3.1 an overview of the various phases, the involved modules and their inputs and outputs are shown.

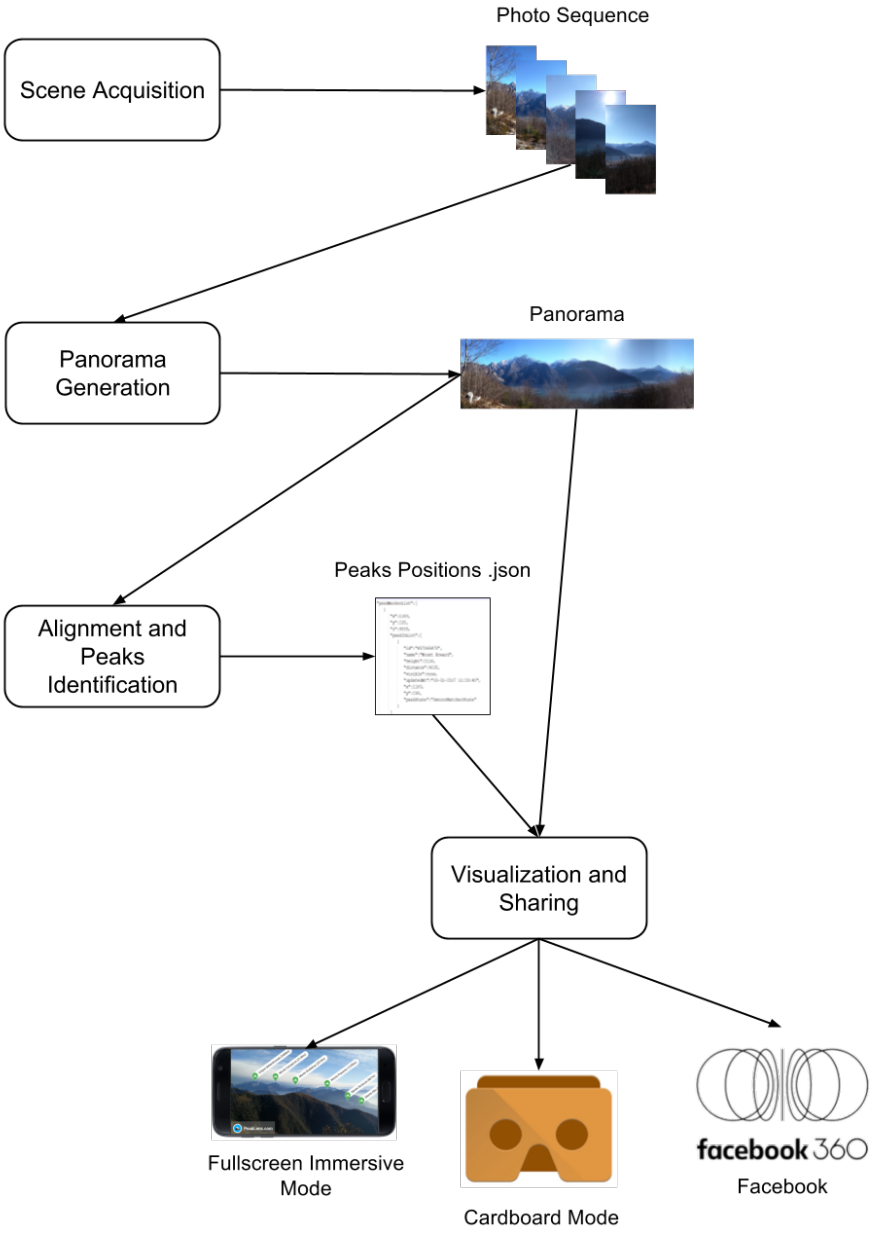


Figure 3.1: Overview of the application modules and phases.

In order to provide a clear overview of the process in Figure 3.1 there is the representation of all and only the main data produced by the application. Both the photo sequence and the panorama are equipped with metadata. Actually the "Alignment and Peak Identification" module uses as additional input some data provided by an external service. Further details on this will be given later. This modular structure has allowed the development and test of each step independently of each other,so future updates and improvements will be easily carried on. The "Scene acquisition" and "Panorama generation" phases are actually consecutive from the user's point of view and he is not aware that these are two different processes. From a conceptual point of view these phases are instead completely decoupled and during the development and test of the application have been treated independently. The first one has been developed and tested directly on the smartphone because it is strictly related to the Android hardware and functionalities. The latter one has been developed and tested on computer and then ported and tested on the smartphone.

3.2 Scene acquisition

In this phase the user can acquire the surrounding environments by shooting some photos using the smartphone. Photos are enriched with latitude, longitude, camera fields of view and orientation metadata. These data are used to monitor the smartphone position in order to guide the user during the shooting and in the next step, in addition to the photo sequence, to compose the panorama. For this purpose a smartphone equipped with gyroscope, accelerometer, magnetometer and gps besides the camera is needed. Although it may sound like a too stringent hardware requirement this is a common scenario for modern smartphones.

3.2.1 Initialization

The first step is the sensor activation and initialization. This is automatically done at the launch of the camera functionality. To work properly the magnetometer needs to be calibrated, so a message that invites the user to move the smartphone around in an eight patterned figure in the air is shown, Figure 3.2. The movement forces the magnetometer to reset, restart and return the correct cardinal direction. The

Figure 3.2: Magnetometer Calibration: how to move the device to help the sensor calibration.

user is also invited to activate the gps by means of a popup. Data on position are mandatory for the following phases, so it is not possible to use the application without activating it. After the calibration and gps activation steps, the interface to shoot photos and a brief tutorial explaining how to move the phone during the acquisition are shown.

3.2.2 Shooting and tilt control

The scene is automatically acquired by rotating the smartphone clockwise around the vertical axis. To control the process and the smartphone orientation, azimuth, pitch and roll are used. The reference coordinate system and the conventions used in this work are shown in Figure 3.3

Considering the device held in the position shown in Figure 3.3, the X axis is horizontal and points to the right, the Z axis is vertical and points up and the Y axis points toward the inside of the screen face (outside the device's camera).

- *Azimuth*: degree of rotation about the z axis. This is the angle between the device's current compass direction and magnetic north. If the screen of the device faces magnetic north, the azimuth is 0 degrees; if it faces south, the azimuth is 180 degrees.

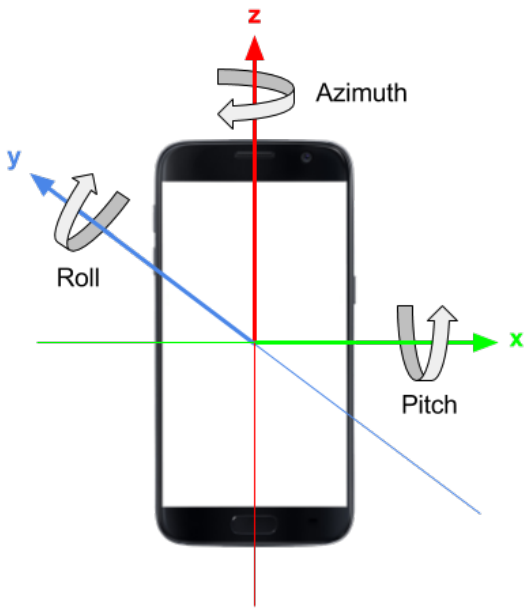


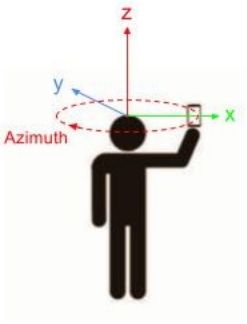
Figure 3.3: The reference coordinate system.

Similarly, it is 90 degrees for east, and 270 degrees for west.

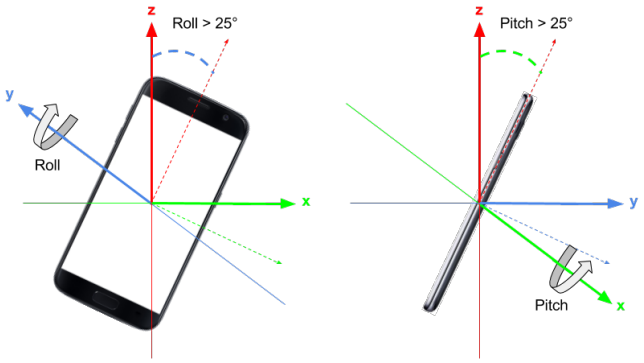
- *Pitch*: degree of rotation about the x axis. This is the angle between a plane perpendicular to the device's screen and a plane parallel to the ground. If the device is perpendicular to the ground with the screen facing the user and the top edge of the device is tilted forward toward the ground, the pitch angle becomes positive. Tilting in the opposite direction causes the pitch angle to become negative. The range of values is -180 degrees to 180 degrees.

- *Roll*: degree of rotation about the y axis. This is the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. If the device is perpendicular to the ground with the bottom edge closest to it and the right edge of the device is tilted toward the ground, the roll angle becomes positive. Tilting in the opposite direction causes the roll angle to become negative. The range of values is -90 degrees to 90 degrees.

Actually two different azimuths are used: the relative and the absolute ones. The *absolute azimuth* follows the convention previously defined and is computed using data provided by magnetometer and accelerometer. This value is required from the "Alignment and peak identification" module for the peak identification phase. The *relative azimuth* is used to control the device orientation during the acquisition as well as to align photos in the "Panorama generation" module. Given the high level of precision needed, data are retrieved using only the gyroscope and the accelerometer. The magnetometer is not used because of inaccuracies caused by magnetic field variations. So, the only difference between the relative and absolute azimuth is that, as it does not use the magnetometer, the relative azimuth reference is not the magnetic north but another one (e.g. the phone position when the sensor is activated). This reference changes across different devices.



(a) Rotation error



(b) Inclination errors

Figure 3.4: Errors during scene acquisition.

As stated before this application generates cylindrical panoramas. The device takes photos while the user rotates it clockwise about the vertical axis. Photos are automatically taken at any 10-degree changes of the relative azimuth. Each photo is saved together with data on location (latitude and longitude), field of view (vertical and horizontal) and orientation (relative azimuth, pitch and roll). Other data such as the absolute azimuth are saved separately and then used as input for the next phase. Ideally the scene should be taken by maintaining the smartphone vertical, without pitch and roll variations and rotating it perfectly around its vertical axis. But this is actually impossible without a tripod. During the acquisition users usually tend to rotate around their vertical axis more than around the smartphone one, Figure 3.4(a). This can lead to parallax errors. Anyway in the next phases these kinds of errors are compensated in a certain measure. For pitch and roll, too,, there is a certain degree of freedom but movements that might compromise the final result are detected, signalled by an error screen and the process is interrupted. If the smartphone is tilted laterally or frontally more than a given threshold the acquisition fails. We have fixed this threshold to 25 degrees, Figure 3.4(b). The acquisition phase finishes when a 360 degree rotation is automatically detected or in any other moments if the user decides to stop it before.

3.3 Panorama generation

After the photo sequence has been captured, the application will start compositing the panorama. The first step is the image alignment with respect to the reference surface chosen for the composition. Later, images are stitched together by applying some different techniques which aim at producing a final panorama without noticeable artifacts. Given the limited amount of memory available on smartphones, both for the alignment and stitching, a sequential approach has been used. This means that at each step only the current image and the panorama are kept in memory.

3.3.1 Image alignment

The alignment process and some techniques studied in literature have been discussed in chapter 2. Now the approach adopted in this work and the reasons underlying the choices made are presented. The fol-

lowing assumptions are at the base of the solution proposed in this section:

- The order of images in a sequence is known a priori
- A fairly accurate estimation of the tilt angles (azimuth, pitch, roll) is available through sensor data
- Camera parameters are known a priori
- The sequence of images is acquired through a rotation around the smartphone vertical axis. Actually there are some inaccuracies to be considered such as those mentioned earlier, Figure 3.4.

This means that there is no need to use techniques to identify the image ordering, estimate camera parameters or the kind of panorama to be composed (spherical, cylindrical etc.). In the proposed algorithm images are first rotated by applying a perspective transformation based on sensors data to compensate tilt errors during the acquisition. Then images are projected onto the cylindrical reference surface and a pixel-based alignment technique is applied to refine the positions estimated through sensors.

Sensor-based alignment

In this step images are sequentially warped by applying a perspective transformation using their pitch and roll values. The first image pitch is taken as reference while the reference roll is set to 0 (phone in vertical position). While the first image is warped by just considering the roll, others are warped considering their rolls and the difference between their pitch and the reference one. By means of the alignment the movement of the camera during the acquisition is reconstructed. Figure 3.5 shows a sequence of four images taken with variable pitch and roll as well as the result after the application of the sensor-based alignment. The algorithm used for the alignment is based on the one proposed in [10].
The goal is to identify and then apply the perspective transform $\mathcal{F}$ to the image:

$$M = \mathcal{F}(w_p, h_p, \theta, \phi, f_v)$$

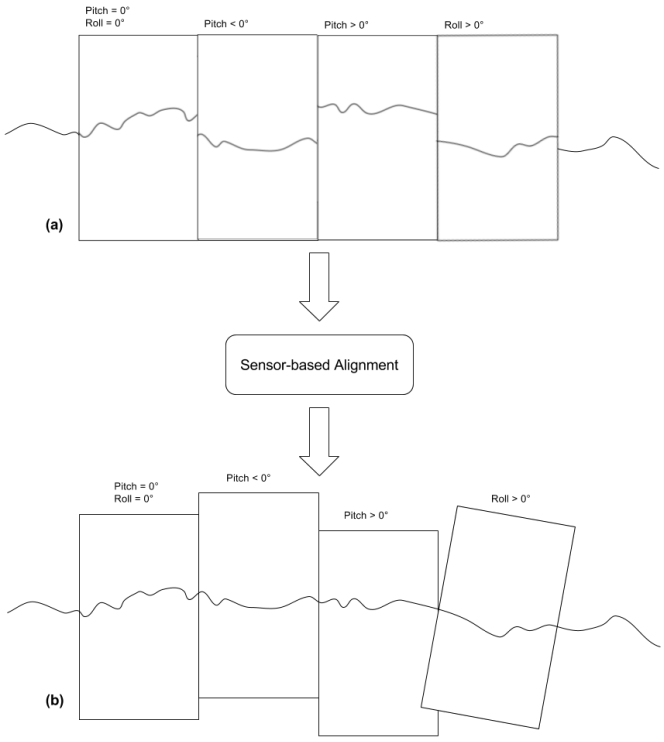


Figure 3.5: Image sequence before and after the application of the sensor-based alignment.

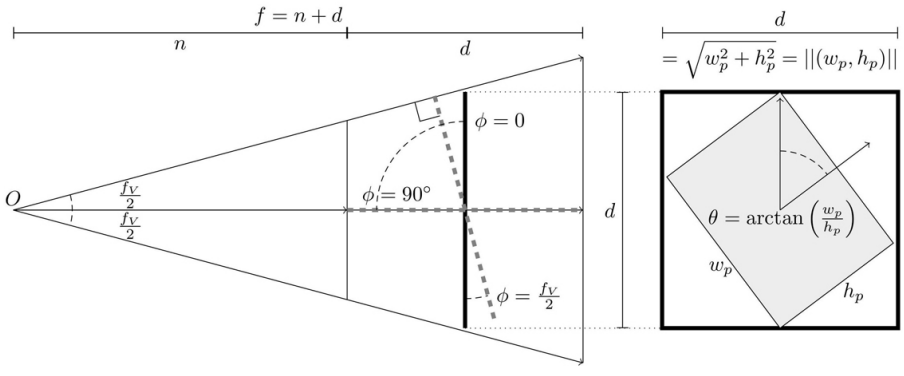


Figure 3.6: Relationship between w_p, h_p, θ, ϕ and f_v.

where w_p, h_p and f_v are the image width, height and vertical field of view, ϕ and θ the pitch and roll variations. Figure 3.6 shows the relationships among these values for the particular configuration ($\phi = f_v/2$ and $\theta = \arctan(w_p/h_p)$) which results in the biggest image to process. The transformation can be defined as:

$$\mathcal{F} = PR_\phi R_\theta T$$

where R_ϕ is the rotation matrix around the x-axis, R_θ the rotation matrix around the z-axis, T the translation matrix that displaces the coordinate system down the z-axis and P is the projection matrix for a square viewport with vertical field of view f_v. The system used for calculations (OpenGL coordinate system) is different from the one used in the acquisition phase Figure 3.3. Its z-axis corresponds to the -y-axis and the y-axis to the z-axis. So, rotations relative to roll variations θ have been applied to the z-axis.

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T defines how much to translate the object down the z-axis. The side length of the square that would contain any rotations of the image is:

$$d = \sqrt{w_p^2 + h_p^2}$$

Given d and f_v the hypotenuse of the right triangle with an opposite side of length $d/2$ subtending an angle $f_v/2$, there follows:

$$h = \frac{d}{2\sin(\frac{f_v}{2})}$$

The matrix T is therefore:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The perspective matrix P is:

$$P = \begin{bmatrix} \frac{n}{n\tan(\frac{f_v}{2})} & 0 & 0 & 0 \\ 0 & \frac{n}{n\tan(\frac{f_v}{2})} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where $n = n - d/2$ is the near plane and $f = h + d/2$ the far plane.

Cylindrical projection

The next step is to project each image onto the cylindrical reference surface.
First a *Forward Warping* is used to map the image onto the cylinder. From image coordinates (x,y), the projected coordinates on the cylinder (x', y') are given by:

$$x^{'} = s\theta = s\tan^{-1}\frac{x}{f}$$

$$y^{'} = sh = s\frac{y}{\sqrt{x^2 + f^2}}$$

Then an *Inverse Warping* is used to map pixels onto the cylinder to the final image pixels. The inverse mapping from cylindrical coordinates (x',y') to the image coordinates (x,y) is:

$$x = f\tan\theta = f\tan\frac{x^{'}}{s}$$

$$y = h\sqrt{x^2 + f^2} = \frac{y^{'}}{s} f\sqrt{1 + \tan^2\frac{x^{'}}{s}} = f\frac{y^{'}}{s}\sec\frac{x^{'}}{s}$$

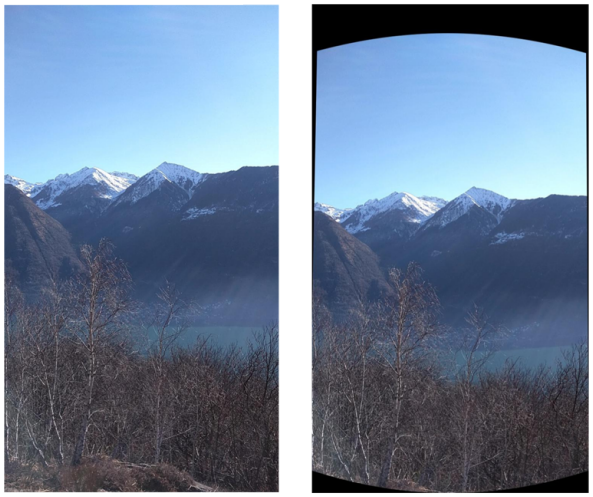


Figure 3.7: Cylindrical projection. a - Original image, b - Image after the sensor-based alignment and cylindrical warping.

Bilinear-interpolation has been used to compute the colors at the destination pixels.

Pixel-based alignment refinement

After the tilt angle compensation performed by the sensor-based alignment algorithm and the cylindrical warping, images are considered related by a translational model. This is actually true for azimuth variations (horizontal translation) but it is an approximation with respect to pitch and roll variations. These variations should be modelled as rotations with respect to the x and y axes Figure 3.3 but, analyzing the sensor-based alignment results, we noticed that pitch and roll tilt angles were quite good compensated in that step. The major problem was caused by the erroneous movement during the acquisition as illustrated in Figure 3.4(a). This introduces in fact a translational component of the movement that cannot be detected by sensors. By means of horizontal translations this component can be managed in this step. Some small residual pitch errors can be compensated by the pixel-based alignment by means of vertical translations. Figure 3.8 shows the stitching result with and without the application of this

technique. In Figure 3.8(b) the problem of repeated objects introduced by the translational component of the azimuth variation is highlighted. Figure 3.8(d) shows how these artifacts disappear after the alignment refinement step.

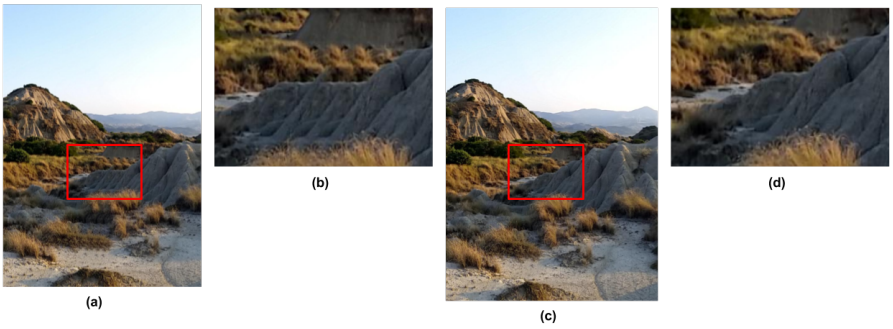


Figure 3.8: Alignment refinement by using the pixel-based technique. a - Image stitched without the application of the pixel-based technique, b - detail of the image showing the introduced artifacts, c - Image stitched after the application of the pixel-based technique, b - detail of the image showing that the introduced artifacts have disappeared.

As stated in chapter 2 the pixel-based alignment consists in a direct comparison of the intensities of the images pixels. The algorithm used is based on the ones presented in [43]. The goal is to compute a horizontal and vertical displacement in order to refine the position of each image. Given two images, at each step, a horizontal and vertical translation is applied to their positions, the error for that offset is evaluated to test how much pixels agree and the process is repeated again. Once all the translations in a given interval have been tried, the one with the minor error is chosen. An error metric to evaluate the quality of the alignment and a search technique to define how to search for a possible solution should be defined.

Windowed sum of squared differences. This is the error metric used to evaluate alignments. Only a given part of the two images overlaps and some pixels may lie outside that. So a weighting function has been used to select only pixels inside image boundaries (this has been implemented as a binary mask).

$$E_{WSSD}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u}) [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2$$

where $I_0(\boldsymbol{x})$ and $I_1(\boldsymbol{x})$ are the two images, $\boldsymbol{x}_i = (x_i, y_i)$ corresponds to pixel locations, $\boldsymbol{u} = (u, v)$ is the displacement and $w_0(\boldsymbol{x})$ and $w_1(\boldsymbol{x})$ are the weighting functions of the two images. These are one inside the valid range and zero outside. The above metric can have a bias towards smaller overlap solutions. To avoid this problem it is divided by the overlap area

$$A = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u})$$

and the *mean squared error* is computed:

$$MSE = \frac{E_{WSSD}}{A}$$

Hierarchical search technique. It is really inefficient and useless to search for all the possible displacements between images. The sensor-based alignment provides in fact a quite good initial approximation of the relative positions. Then the search is performed in a fixed interval and a hierarchical method is used to speed up the process. An image pyramid of n levels is constructed and a search over a small number of pixels is first performed at coarser levels [43]. The position estimated from one level is then used to initialize another search at the next finer level. At each new level l the image is generated by smoothing and halving, by subsampling pixels, the image at level $l - 1$.

$$I_k^{(l)}(\boldsymbol{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\boldsymbol{x}_j)$$

At the coarser level l a full-search for the best displacement is performed over a given range:

$$\boldsymbol{u}^{(l)} \in 2^{-l}[-S, S]^2$$

where S is the desired search range at the finest (original resolution image) level. The resulting displacement $\boldsymbol{u}$ is then used as initial displacement to initialize the search at the next finer level.

$$\hat{\boldsymbol{u}}^{(l-1)} \leftarrow 2\boldsymbol{u}^{(l)}$$

At the end vertical translations are applied to each image and their azimuth value is updated with respect to the detected horizontal displacement . Images are saved locally and the stitching phase can begin.

3.3.2 Image stitching

In this stage images are stitched together to compose the final panorama. First, image colors are equalized, then the best path to stitch each image pair is identified and finally a blending technique is used to seamlessly stitch images.

Color Correction

Figure 3.9: Stitching without color correction.

Figure 3.10: Stitching using color correction.

Color correction aims at correcting the color differences between images due to the use of automatic settings for exposure and white balance during the acquisition. In PeakLensVR users can decide whether to

use automatic settings or not. In the second case this phase can be skipped. The color correction has been moved before the pixel-based alignment in the final application implementation. As a matter of fact, since the alignment refinement is based on pixel intensities, the use of equalized pixel values will produce better results. The implemented algorithm is based on the one presented in [48]. For all images color correction coefficients are computed. To match the colors, linearized RGB values have been used instead of the default gamma-corrected RGB. A detailed explanation of differences between these is provided in [23]. In an image sequence $S_0, S_1, ...S_i, ...S_n$ given that S_{i-1} and S_i are adjacent images and S^o_{i-1} and S^o_i where they overlap, the correction coefficient for the image S_i is:

$$\alpha_{c,i} = \frac{\sum_p (P_{c,i-1}(p))^\gamma}{\sum_p (P_{c,i}(p))^\gamma} \quad c \in \{R, G, B\}\ (i = 1, 2, 3, ..., n)$$

where $P_{c,i-1}(p))$ is the color value of pixel p in image S^o_{i-1}; $P_{c,i}(p))$ is the color value of pixel p in image S^o_i; $\gamma = 2.2$ is the gamma-correction coefficient. The color correction coefficient of the first image $\alpha_{c,0}$ is set to 1. By multiplying pixel values of the image S_i with the coefficient $\alpha_{c,i}$ the image S_i color is equalized to the one of the image S_{i-1}. Viceversa the image S_{i-1} color is equalized to the one of the image S_i by multiplying its pixel values with $1/\alpha_{c,i}$. To avoid color saturations a global adjustment factor g_c for the whole image sequence is computed, so that the overall adjustments $g_c\alpha_{c,i}$ approximate 1, by solving the least-squares equation

$$\min_{g_c} \sum_{i=0}^{n} (g_c\alpha_{c,i} - 1)^2 \quad c \in \{R, G, B\}$$

This equation can be solved in closed form by setting the derivative to 0,

$$g_c = \frac{\sum_{i=0}^{n} \alpha_{c,i}}{\sum_{i=0}^{n} \alpha_{c,i}^2} \quad c \in \{R, G, B\}\ (i = 0, 1, ..., n)$$

In order to apply the color correction an image (the best image) should be used as reference image to correct the others. It is difficult to automatically determine the image with the best colors. As a heuristic

the image with the median *Luma* value has been selected. The *Luma* represents the brightness of an image and is computed by the weighted sum of gamma-corrected RGB components [13]:

$$Y^{'} = 0.2126R^{'} + 0.7152G^{'} + 0.0722B^{'}$$

where R'G'B' are the mean gamma-corrected values of the image red, green and blue channels. Color correction coefficients are computed at the launch of the stitching process. Images are then corrected sequentially at each iteration. The correction is applied by multiplying the image with a new color correction coefficient computed by multiplying color correction coefficients between the best image and the one to be corrected, if the image is after the best image in the sequence

$$\alpha_{c,i}^{'} = (\prod_{j=k+1}^{i} \alpha_{c,j} g_c)^{\frac{1}{\gamma}}$$

where k is the position of the best image in the sequence. Or, if the image is before the best image, by multiplying the inverse of the correction coefficients

$$\alpha_{c,i}^{'} = (\prod_{j=i+1}^{k} \frac{1}{\alpha_{c,j} g_c})^{\frac{1}{\gamma}}$$

Coefficients are raised to $1/\gamma$ to go back from linearized to gamma-corrected RGB values.

Figure 3.9 shows the result of stitching an image sequence acquired using automatic settings without the application of the color correction algorithm. The variation of color is evident across the image. Figure 3.10 shows the same image sequence stitched by using the proposed algorithm.

Seam estimation

Moving objects and residual alignment errors may cause ghosting artifacts in the final panorama. Seam estimation is used to find the best path to cut images in the overlapping area of the source images to be

Figure 3.11: Example of an optimal seam estimated in the overlapping area of two images.

stitched. The goal is to cut and then merge images in places where they differ the least. Like in [48],[33] and [34] dynamic programming is used to find the optimal seam. As stated in the previous chapter dynamic programming is quick and uses less memory compared with other approaches (e.g graph cut techniques). So it is suitable to a mobile implementation. Given two images I_i and I_j and their overlapping areas I_i^o and I_j^o an error surface is computed, Figure 3.12 (a)

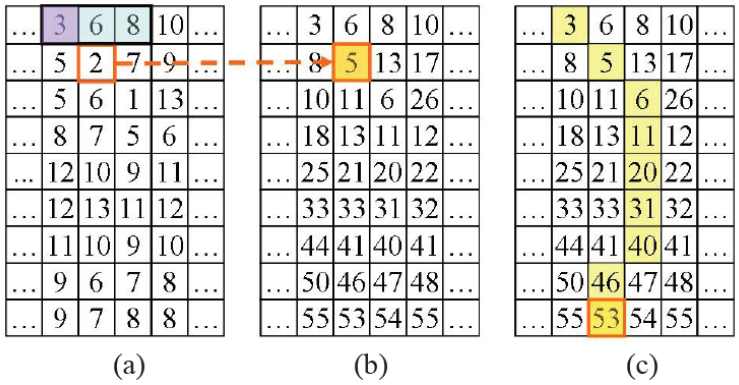


Figure 3.12: Optimal seam estimation by using dynamic programing. (a) $C(x, y)$, (b) $C'(x, y)$, (c) minimum path.

$$C(x, y) = (I_i^o(x, y) - I_j^o(x, y))^2$$

Then, costs are updated starting from the error surface second row by accumulating them with the minimum cost from the upper row, Figure 3.12 (b)

$$C'(x, y) = C(x, y) + \min[C'(x-1, y-1) + C'(x, y-1) + C'(x+1, y-1)]$$

Finally, the minimum cost path is traced from the last row to the first. The optimal x position x_{opt} is identified for each row y, Figure 3.12 (c)

$$x_{opt}^y = \min_{x \in S} C'(x, y), \quad S = \{x_{opt}^{y+1} - 1, x_{opt}^{y+1}, x_{opt}^{y+1} + 1\}$$

The optimal x position is restricted within 3 pixels from the optimal position of the row $y + 1$.

MVC blending

After the best seam has been found, the right part of the image is stitched to the right of the current panorama using a blending technique to smoothen the transition. The first blending technique we tried was a simple Linear Blending [48]. This is a very simple and light approach but results are compromised by moving objects, alignment errors, and differences between images along the seam. A powerful and effective approach that performs image blending in the gradient domain is Poisson Blending [47]. However it needs quite high computational and memory resources.

The solution finally adopted has been the Mean Value Coordinates Blending. This is based on the approach proposed in [26] for instant image cloning and used in [48] for stitching purposes. Results have proved to be as good as the ones produced by using the Poisson Blending with lower computationally and memory requirements. Given an image S_i to be blended to the current panoramic image I_p, the color values of the points $p_1, p_2, ..., p_n$ on the seam m_c in the panoramic image I_p are taken as reference to correct pixel colors in the image S_i, so that no visible seam remains. This is done by first computing color differences along the seam between pixels in I_p and pixels in S_i and then distributing and adding color differences to the rest of S_i, Figure 3.13. Let $P_1, P_2, ..., P_n$ be those differences and q be a pixel on the image S_i, the value to be added to q to correct its color is computed by interpolating the $P_1, P_2, ..., P_n$ values, Figure 3.14 (b):

$$P(q) = \sum_{i=1}^{n} w_i(q) P(p_i)$$

where the weights are the inverse coordinate distances to the boundary pixels, normalized:

$$w_i(q) = \frac{1/||p_i - q||}{\sum_{j=1}^{n} 1/||p_i - q||}$$

Finally for each pixel $q \in S_i$ the final color is $Q = q + P(q)$.

To speedup the computation an optimization method has been used. Pixel correction colors are accurately computed only at pixels increasingly far away from the seam. After, these values are (bi)linearly in-

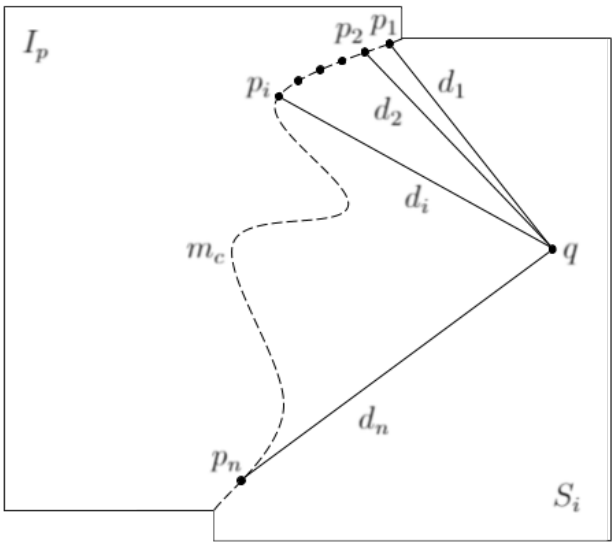


Figure 3.13: MVC Blending.

terpolated for other pixels. To select pixels in which to compute the correction values $P(q)$ a *Quadtree* [17] has been used, Figure 3.14 (a).

3.4 Alignment and peak identification

The goal of this phase is the identification of the mountain peaks in the generated panoramas. The first step is the identification of peak positions by only using sensors' data. This initial estimation is then refined by performing a comparison between the mountain skyline extracted from the image and a reference skyline extracted from a computer generated 3D model of the earth. The algorithm used in this module has been developed in the SnowWatch and PeakLens projects and adapted to the case of panoramic images. In chapter 1 an introduction to these projects and the technologies developed has been stated. This section describes how these technologies have been used in the context of the PeakLensVR application. This module is completely decoupled from the others and it needs an Internet connection to work. So, if the connection is available at the end of the panorama generation phase, the module is automatically launched, otherwise the panorama is placed in

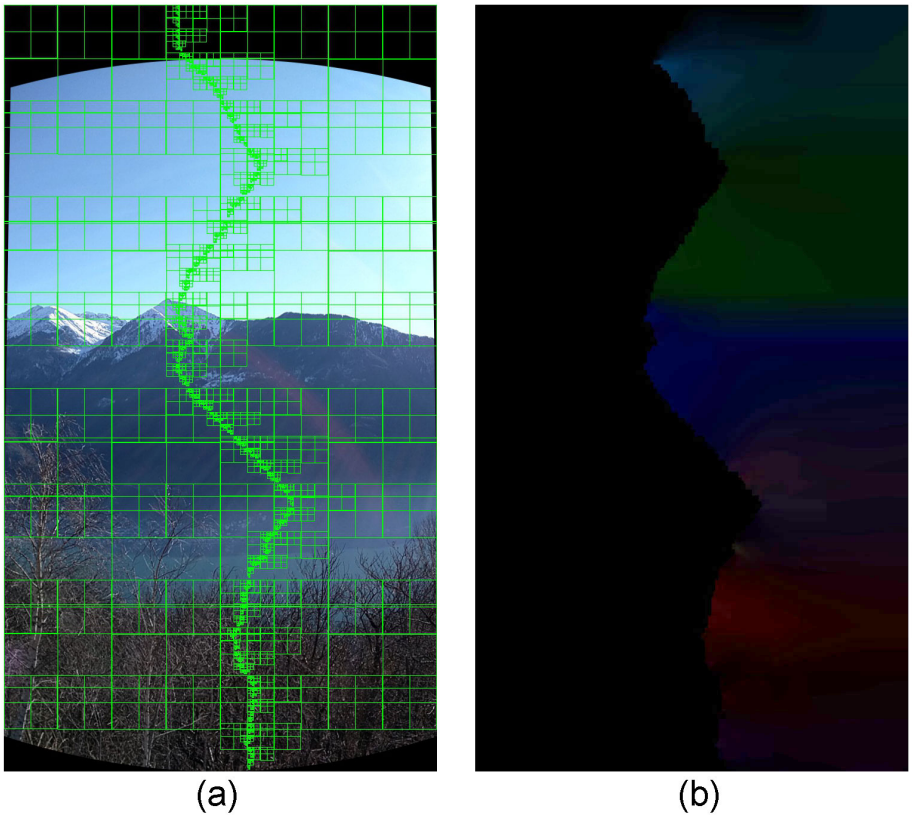


Figure 3.14: (a) quadtree representation, (b) P(q) values computed interpolating color differences on the seam between the panorama and the image to be blended.

a waiting list. At a future launch of the application, if the connection is available, this phase will be sequentially executed for all the waiting panoramas.

3.4.1 Sensor-based peak positioning

In this step an external service developed during the PeakLens project has been used. This is an online service that, given latitude and longitude coordinates and some others data, returns an image with the representation of the skyline visible from that point as well as the list of peaks and their positions with respect to the skyline. For the Sensor-based peak positioning, peak coordinates are simply remapped from the Skyline Frame (the skyline image coordinate system XOY) to the Panorama Frame (the panoramic image coordinate system xoy).

The skyline image I_s, returned from the online service, has width w_s, height h_s, horizontal field of view $hFOV_s = 360$, vertical field of view $vFOV_s = 180$, $pitch_s = 0$, $roll_s = 0$ and $azimuth_s = 0$ (North) at the skyline image left border ($X = 0$). The panoramic image I_p has width w_p, height h_p, horizontal field of view $hFOV_p$, vertical field of view $vFOV_p$, $pitch_p$, $roll_p = 0$ and the $azimuth_p$ value is referred to the image center ($x = w_p/2$). The steps to remap peak positions $P(P_x, P_y)$ from the Skyline Frame to the Panorama Frame are the following:

$$w_{pscaled} = hFOV_p s_s, \quad h_{pscaled} = vFOV_p s_s$$

to scale the Panorama Frame with respect to the Skyline Frame, where $s_s = w_s/hFOV_s$ is the Skyline Frame pixel-degree ratio.

$$X_o = X_c - \frac{w_{pscaled}}{2}, \quad \text{if } X_o < 0 \text{ then } X_o = X_o + w_s$$
$$Y_o = Y_c - \frac{h_{pscaled}}{2}$$

to compute the Panorama Frame origin with respect to the Skyline Frame.

$$X_c = azimuth_p s_s, \quad Y_c = \frac{h_s}{2} + pitch_p s_s$$

are the coordinates of the panoramic image center into the Skyline Frame.

$$P_x^{'} = P_x - X_o, \quad \text{if } P_x^{'} < 0 \text{ then } P_x^{'} = P_x^{'} + w_s$$
$$P_y^{'} = P_y - Y_o$$

to compute peak positions with respect to the Panorama Frame scaled into the Skyline Frame.

$$P_x^{''} = P_x^{'} \frac{w_p}{w_{pscaled}}, \quad P_y^{''} = P_y^{'} \frac{h_p}{h_{pscaled}}$$

to compute the final peak positions in the Panorama Frame. Finally the list of remapped peaks will contain only peaks with $0 \leq P_x^{''} < w_p$ and $0 \leq P_y^{''} < h_p$.

3.4.2 Global Matching and position refinement

Positions estimated by using sensor data might contain imprecisions due to sensor errors. In PeakLens and SnowWatch projects some algorithms to refine these positions, by means of a deep analysis of the image, have been developed. The first step is the extraction of the skyline representation from the panoramic image. During years a lot of work has been done to improve the precision of the extraction algorithm. This is now capable to distinguish the skyline from other elements of the photo such as trees or clouds, even in case of partial occlusions caused by objects such as buildings. In [27] the extraction algorithm is explained in detail. The skyline image extracted from the panorama is then scaled with respect to the skyline image downloaded from the online service and its initial position in the Skyline Frame is computed (same process of Panorama Frame - Skyline Frame mapping). From that position the global matching algorithm is applied to find the best position with respect to the reference skyline. The best position is considered to be the one in which the 2 skylines overlap more. This is found by applying some vertical and horizontal displacements in a given interval and computing a score for each position. The score is based on the degree of overlap of the two skylines. Finally, the position with the highest score is returned as an offset with respect to the initial position and it is added to peak positions. Positions computed in this step should be closer to the actual peak positions than the ones computed by only using sensor data. Results will be analyzed and evaluated in chapter 6 and the factors that influence the effectiveness of this method Will be discussed. In Figure 3.15 (a) the positioning of the panorama in the Skyline Frame by using sensor data only is Shown. Figure 3.15 (b) shows the result after the application of the global matching algorithm. The positioning in (b) is clearly better. In both figures the skyline extracted from the panorama is highlighted in red and in (b) the green color has been used to highlight pieces that overlap with the reference skyline.

3.5 Visualization and sharing

Users have different ways to visualize and share the multimedia content produced by the application. They can see panoramas in an immersive environment using the Cardboard mode by means of a VR headset or

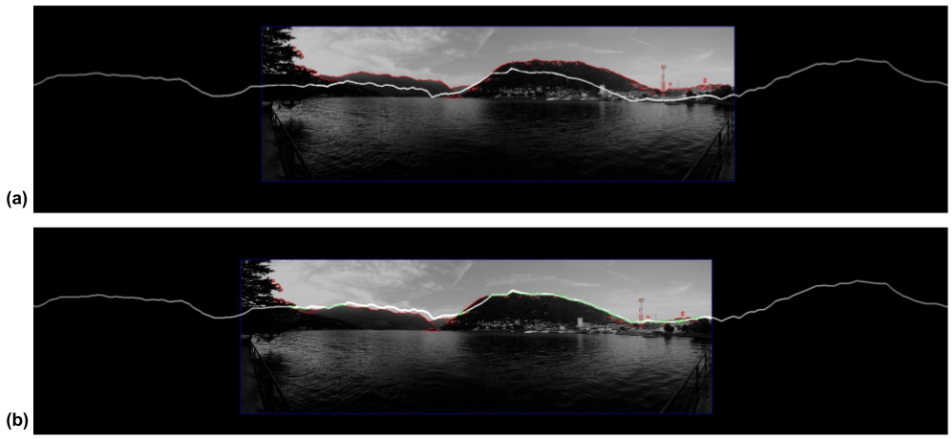


Figure 3.15: (a) panorama aligned with the reference skyline by using sensor data, (b) alignment refinement after the global matching algorithm.

in the Fullscreen immersive mode without additional hardware. It is possible to share panoramas on Facebook or with other users that use the PeakLensVR application.

3.5.1 Cardboard and Fullscreen immersive mode visualization

In both *Cardboard* and *Fullscreen immersive* modes the digital environment is built by projecting the panorama and the peaks onto a cylindrical surface. The camera that generates the user view is placed at the center of this cylinder. In *Cardboard mode* it is possible to turn around and watch the scene by moving the head as if looking at a real panorama. Sensors capture movements and apply them to the camera inside the 3D scene automatically changing the user view. Furthermore some hotspots users can interact with are placed in peak positions. It is possible to point at the hotspot by looking at it and click by means of a controller to visualize a 3D dialog with all the available information for that peak. In *Fullscreen immersive mode* instead of clickable hotspots some labels with peak names are shown. It is possible to click and drag the panorama to turn around in the scene. In order to build the 3D environment the main task is the mapping of the panorama inside the 3D scene and the transformation of peak coordinates from the planar coordinate system used for the panoramic image to the cylindrical reference system used for the scene. Panoramas generated with the

application have a horizontal field of view less or equal to 360 degrees. The cylinder used to build the 3D scene always represents a complete 360 degree panorama. So to generate the 3D scene, first the original panorama and peaks are mapped to another planar reference frame that represents a complete 360 degree panorama, Figure 3.16. Then the new image is mapped to the cylindrical reference frame, Figure 3.17.

From Panorama to Panorama360 reference Frame

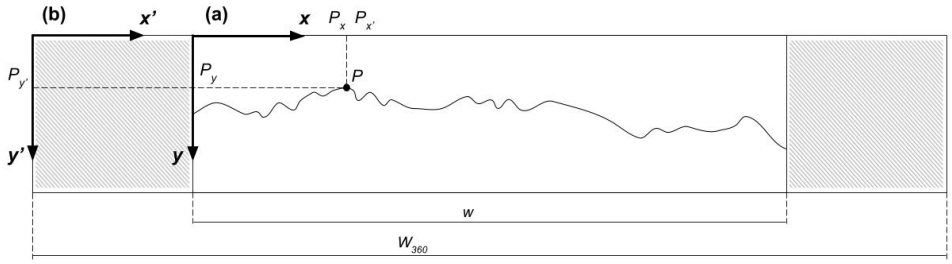


Figure 3.16: Mapping from Panorama to Panorama360 Frame.

Given a panoramic image of width w, height h, horizontal field of view $hFOV$ and vertical field of view $vFOV$, if $hFOV < 360$ then a new panorama with $hFOV = 360$ is built. The image is placed into the new panorama reference systems and also peak positions are remapped.

$$W_{360} = \frac{w}{hFOV} 360\,, \quad \text{width of the 360 degree panorama.}$$

With $hFOV_{360} = 360$, $vFOV_{360} = vFOV$, $h_{360} = h$.

The image is placed in the new reference frame so that its center coincides with the center of the Panorama360. Its top left corner in the new frame will be:

$$x_0^{'} = \frac{W_{360} - w}{2}\,, \quad y_0^{'} = 0$$

Then peaks positions are remapped:

$$P_x^{'} = P_x + x_0^{'}\,, \quad P_y^{'} = P_y$$

From Panorama360 to the 3D Scene Frame

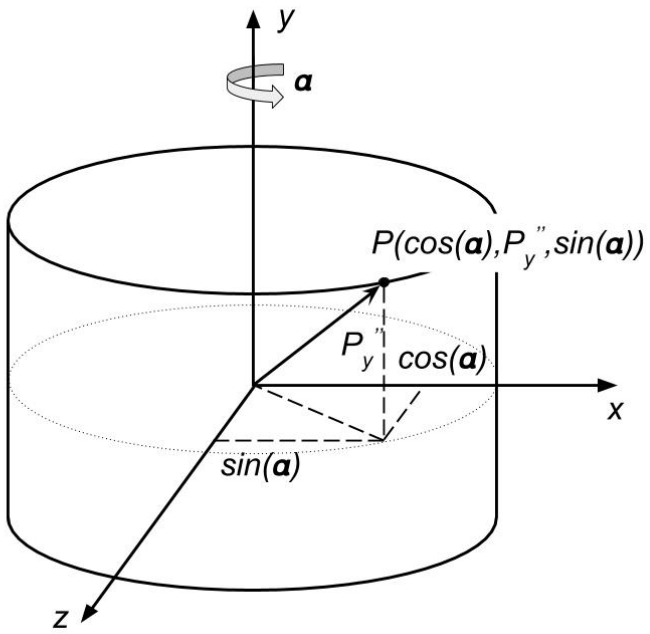


Figure 3.17: Cylindrical coordinate reference system used to build the 3D scene.

The 3D Scene is built by wrapping the Panorama360 into a cylinder and remapping peaks to the new cylindrical reference system. The new peak coordinates will become:

$$P_{cyl} = (x, y, z) = (\cos(\alpha), P_y^{''}, \sin(\alpha))$$

Theese coordinates refer to the 3D Scene Frame shown in Figure 3.18 (b). The reference system adopted by the library used to implement this task (OpenGL) is the one shown in Figure 3.18 (a). The zero position in system (b) differs 90 degrees clockwise with respect to system (a) and the direction of rotation is inverted. So α is:

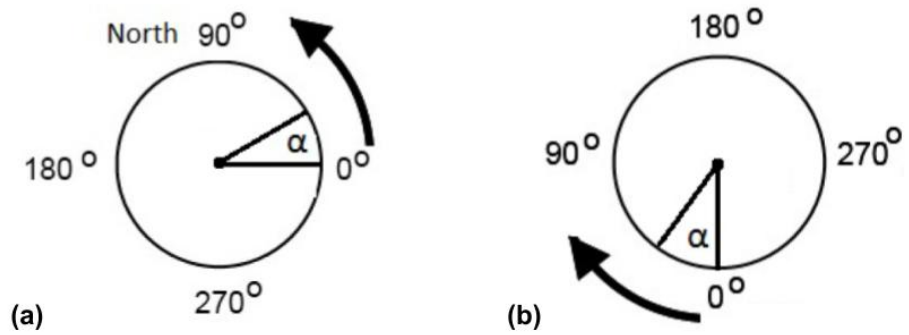


Figure 3.18: The Calculation Reference System (OpenGL) (a) and the 3D Scene Frame (b).

$$\alpha = -\alpha_0 - \frac{\pi}{2}$$

α_0 is the rotation computed in the 3D Scene Frame and it is:

$$\alpha_0 = \frac{hFOV_{360}}{W_{360}} P'_x$$

Given that the cylinder representing the scene is centered in the coordinate reference system, there follows:

$$P''_y = -(P'_y - \frac{h_{360}}{2})$$

Peak Clustering

Sometimes it might happen that some peaks are so close that hotspots rendered in the 3D scene overlap. To avoid this problem the peak list is rearranged before the visualization phase, grouping the too close peaks into clusters. In Cardboard mode users can interact with the clusters by clicking on them and visualizing the list of peaks clustered in that position. To create clusters a *Hierarchical Agglomerative Method* has been used [9]. The *Agglomerative Method* is a bottom-up approach: each peak starts as a single cluster and pairs of clusters are merged moving up the hierarchy. In order to decide if two sets should be

combined, the clustering process needs the definition of a *Dissimilarity Measure*. The *Dissimilarity Measure* consists in the specification of the metric to be used to compute the distance between sets and of a linkage criterion. Sets with a distance less than a given distance are grouped into clusters. When in two sets there is more than one element the linkage criterion is used to determine how to compute the distance between them (e.g. complete-linkage clustering uses the max distance between two elements in different sets). In our case the Euclidean distance has been used as metric and the Centroid linkage clustering (the distance between cluster centroids) as linkage criterion. In the context of this application a 2-level hierarchy is enough. So at the end of the clustering process the list is organized as a list of clusters in which clusters are composed of one or more peaks.

3.5.2 Sharing

PeakLensVR panoramas can be shared among users that use the application. All the metadata needed to exploit the peak identification service are embedded inside the image. Users just need to put the panorama received from another user inside the application folder. At the application launch the new panorama is detected and the service to identify the peaks is automatically invoked. Once the list of peaks has been generated, the user can visualize the image as if he had taken it.

Panoramas can also be shared on Facebook and visualized in the Immersive mode provided by the social network. The application uses in fact the same Facebook metadata (Photo Sphere XMP) to allow the rendering of them in the immersive environment.

Chapter 4

Implementation Details

In this chapter the implementation of the whole PeakLensVR application is presented. An overview of the implementation architecture is given, then the application modules and functionalities are analyzed in detail.

4.1 Overview of the implementation architecture

The application has been developed for the Android platform and the adopted programming language is Java. PeakLensVR is composed of different modules with very specific functionalities. Each module can be executed separately from others and can be seen as an autonomous system. For each execution phase presented in the previous chapter a software module has been developed. Figure 4.1 shows an overview of the application architecture and the main modules implemented.

- *Scene acquisition:* this is the module more related to the Android environment. In fact it uses some Android API (Sensors, Camera, Location) for the acquisition of photo sequences with sensor data for the following generation of panoramas and identification of peaks.
- *Panorama generation:* this consists in the implementation of all the image processing algorithm for the generation of panoramic images that has been presented in the previous chapter. It has been developed separately (on desktop environment for ease of development and testing) and then imported into the main ap-

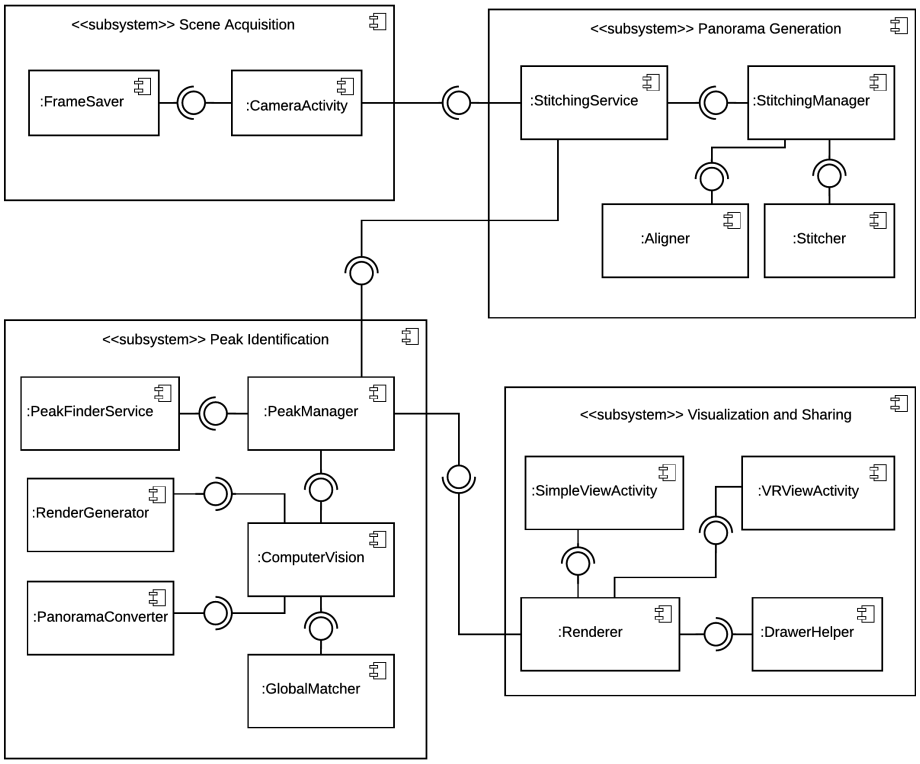


Figure 4.1: Overview of the application architecture and the main modules implemented

plication as an external module. It has been developed in Java language using the Java version of the OpenCV library. OpenCV is a wide library which provides a series of API for computer vision and image processing tasks. In this module it has been used for all the processing tasks on images.

- *Alignment and peak identification:* this is the module that has been imported from the PeakLens project. It is implemented in Java and it also uses the OpenCV library for image processing tasks. It has been adapted to work with very wide field of view images. The scope of this module is the identification of the peak present in images both using sensor data and the global matching algorithm developed in PeakLens.

- *Visualization and sharing:* all the classes and methods used for rendering the 3D scene are contained in this module. It uses the

OpenGL ES 2.0 library for rendering 3D graphics and the Google VR SDK for all the tasks related to the stereoscopic rendering and the interaction with the virtual environment. It also manages the sharing of panoramas outside the application.

4.2 Scene acquisition

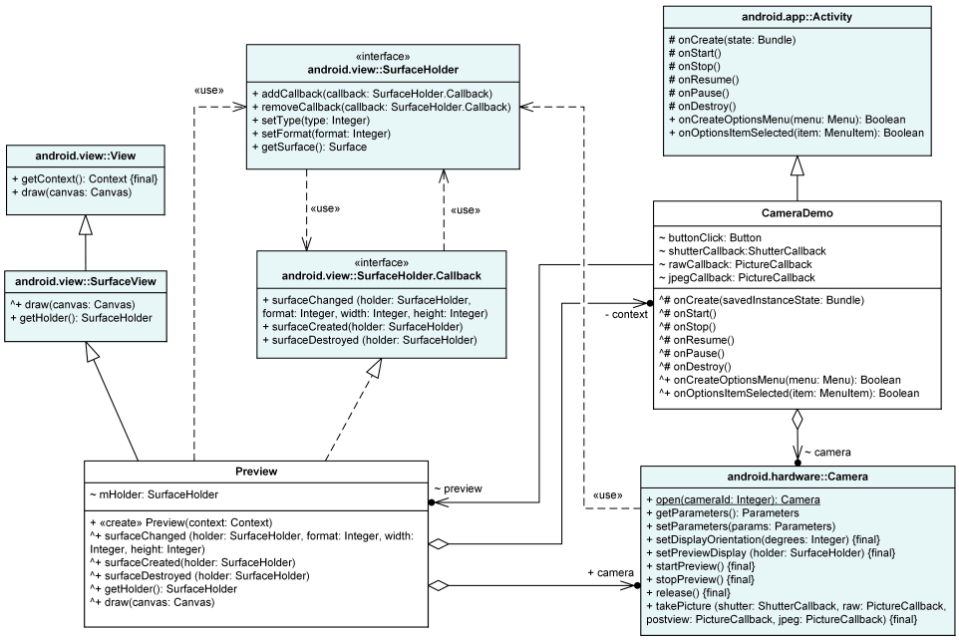


Figure 4.2: Standard Android Camera implementation class diagram.

The figure 4.2 taken from [3] shows the classes needed for the implementation of a standard camera functionality using Android Camera API. This implementation enables the application to receive raw frame data from the camera, to show frames on the screen and to save a frame when a user decides. The *CameraDemo* (which in this application is named *CameraActivity*) extends the *Activity* class, the Android class (which is in charge of managing the creation of the user interface) as well as the interaction with the user. In addition it manages all the other tasks related to the acquisition functionality. By means of the *SurfaceHolder* and *SurfaceHolder.Callback*, the *Preview* class is in charge of the creation and management of the surface to display frames on screen. It also initializes *Camera* parameters. The *Cam-*

era class is the one used to control the smartphone camera (e.g. start or stop preview, take photos). The PeakLensVR implementation differs from the one in Figure 4.2 in the way in which frames are saved. There is no need for a direct user interaction to select the frame to be saved. Users indirectly perform the acquisition by rotating the phone but without being aware of the process. The application automatically decides which frame to save by using data received from sensors. These data are used to manage the acquisition as well as to control the smartphone position during the process to avoid erroneous movements. All frames are saved with metadata on orientation, geographical coordinates and some camera parameters such as the field of view. The acquisition process is divided in two phases (Figure 4.3): initialization and shooting and tilt control.

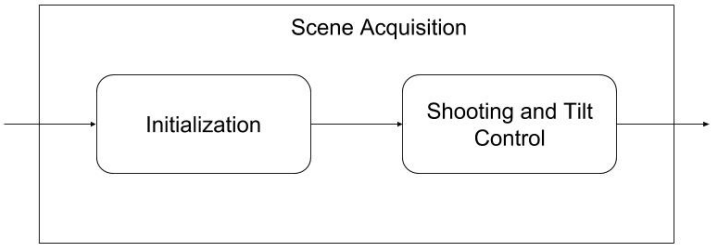


Figure 4.3: Scene acquisition main phases.

4.2.1 Initialization

Some different sensors are used to provide data needed by the application. Table 4.1 provides an overview of these kinds of sensors, the data produced and the Android API used. Except the classes and interfaces used for the GPS data, which are contained in the *android.location* package, other classes are all inside the *android.hardware* package. Android provides some software sensors that generate data by processing data from one or several physical sensors. In Table 4.2 the ones used in this application are listed. The game rotation vector provides more precise data than the rotation vector because it is not influenced by the magnetic field. This is used to control the smartphone orientation during the acquisition while the second is used to save the absolute position of the panorama with respect to the north pole (used in the

peak identification phase).

Data	Sensor	API
Abs.Azimuth	Rotation Vector	Sensor SensorEvent SensorEventListener SensorManager
Rel.Azimuth, Pitch, Roll	Game Rotation Vector	
Latitude, Longitude	GPS	Location LocationListener LocationManager
hFOV, vFOV	Camera	Camera

Table 4.1: Sensors, data and Android API classes used during the scene acquisition.

Software Sensor	Physical Sensors	Description
Rotation Vector	Accelerometer Magnetometer (Gyroscope)	Azimuth variation with respect to the North pole
Game Rotation Vector	Accelerometer Gyroscope	Variations with respect to the smartphone initial position

Table 4.2: Software and hardware sensors used to check the smartphone position and orientation.

In addition to the tasks strictly related to the photo acquisition the *CameraActivity* is in charge of managing, initializing and retrieving data from sensors. Field of view data are directly available in the *Camera* class after its instantiation. To start using sensors listed in Table 4.2 it is necessary to create an instance of the *SensorManager* class and by using it to instantiate the specific sensor objects. Finally to start receiving data it is necessary to register a listener for each sensor. Listeners should be unregistered as soon as they are no longer needed or when application is paused or stopped to save battery. The steps needed for the GPS sensor are almost the same. Listing 4.1 shows how these steps have been implemented inside the *CameraActivity* class.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_camera);
    Log.d("log", "onCreate");

    calibrationFragment = new CalibrationFragment();
    showCalibrationFragment();

    //Sensor Manager
    sensorManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);

    //Relative position
    if (sensorManager.getDefaultSensor
    (Sensor.TYPE_GAME_ROTATION_VECTOR) != null) {
        gameRotationVector = sensorManager.getDefaultSensor
        (Sensor.TYPE_GAME_ROTATION_VECTOR);
    }

    //Absolute Position
    if (sensorManager.getDefaultSensor (Sensor.TYPE_ROTATION_VECTOR) !=
    null) {
        rotationVector =
        sensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
    }

    //Location Manager
    locationManager = (LocationManager)
    this.getSystemService(Context.LOCATION_SERVICE);

    //UI
    initUI();
}

@Override
protected void onPause() {
    super.onPause();
    ...
    // Initialize sensors
    initSensors();
    ...
}

@Override
protected void onPause() {
    super.onPause();
    ...
```

```
43      //Release sensors resources
        releaseSensors();
45      ...
    }
47
    private void initSensors() {
49      //Position
        sensorManager.registerListener(this, gameRotationVector,
        SensorManager.SENSOR_DELAY_FASTEST);
51      sensorManager.registerListener(this, rotationVector,
        SensorManager.SENSOR_DELAY_FASTEST);

53      //Location
        if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION) !=
        PackageManager.PERMISSION_GRANTED) {
55          return;
        }
57      locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,
        0, locationListener);
        }
59
    private void releaseSensors() {
61      //Position
        sensorManager.unregisterListener(this, gameRotationVector);
63      sensorManager.unregisterListener(this, rotationVector);

65      //Location
        if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION) !=
        PackageManager.PERMISSION_GRANTED) {
67          return;
        }
69      locationManager.removeUpdates(locationListener);
    }
```

Listing 4.1: Sensor instantiation, registration and initialization.

The *CameraActivity* implements the *SensorEventListener* interface so it registers itself as listener. The rotation vector sensor due to the usage of the magnetometer needs to be calibrated. The calibration is managed by the *CalibrationFragment* and its instantiation is shown in Listing 4.1 rows 7-8. After the sensor initialization the *CameraActivity* initializes the UI and waits for calibration and data returned from the GPS. These data are mandatory and so users cannot start shooting until they are available.

4.2.2 Shooting and tilt control

As soon as the user clicks the button to start capturing the scene, the location and rotation vector listeners are unregistered (to save battery). The game rotation vector still remains to control the acquisition process. The *CameraActivity* uses some Boolean variables together with the sensor data to decide when to save frames. The *Camera* class *setPreviewCallback()* method enables the installation of a callback invoked for every frame received from the camera besides showing it on screen. So, a callback is instantiated to save frames in background without the need to call the *takePicture()* method. The implemented callback is shown in Listing 4.2.

```
Camera.PreviewCallback previewCallback = new Camera.PreviewCallback() {
    @Override
    public void onPreviewFrame(byte[] data, Camera camera) {

        //If a frame can be acquired
        if (canAcquire) {
            canAcquire = false;
            frameNumber++;

            //Put the frame in the buffer
            try {
                panoFramesBuffer.put(new PanoFrame(frameNumber,
                orientationInDegree.clone(), data));
                if (stopAndSave) {
                    stopCaptureAndSave();
                }
            } catch (InterruptedException e) {
                frameNumber--;
                Log.e(TAG, e.getMessage());
            }
        }
    }
};
```

Listing 4.2: Preview callback used to automatically save frames.

The Boolean variable *canAcquire* is used to lock/unlock the acquisition. It is set to true basing on sensor data. Once true, the frame is saved and it is set to false again. The *onSensorChanged()* method defined in the *SensorEventListener* interface and overridden by the *CameraActivity* is the one that receives sensor data and that sets *canAcquire* to true. The *SensorEvent* is the class that holds information produced

by sensors. Listing 4.3 shows how sensor data are used to control the frame acquisition as well as to check erroneous positions.

```
@Override
public void onSensorChanged(SensorEvent event) {

  //If the event is generated from the game rotation vector (relative movement)
  if (event.sensor.getType() == Sensor.TYPE_GAME_ROTATION_VECTOR) {

    //Remap coordinates to the appropriate coordinate system
    SensorManager.getRotationMatrixFromVector(rotationMatrix, event.values);
    SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.AXIS_X,
    SensorManager.AXIS_Z, rotationMatrixRemap);

    // Transform rotation matrix in azimuth/pitch/roll
    SensorManager.getOrientation(rotationMatrixRemap, orientation);

    //Convert azimuth to degree
    float azimuth = orientation[0] * RAD_TO_DEGREE;
    //Change from -180-+180 to 0-360
    if (azimuth < 0) azimuth += 360;

    if (resetted) {
      referenceRelativeAzimuth = azimuth;
      resetted = false;
    }

    //Adjust with respect to the reference value (acquisition phase)
    azimuth = azimuth - referenceRelativeAzimuth;
    if (azimuth < 0) azimuth += 360;

    //Convert pitch and roll to degree
    float pitch = orientation[1] * RAD_TO_DEGREE;
    float roll = orientation[2] * RAD_TO_DEGREE;

    //To access globally to the adjusted and converted values
    orientationInDegree[0] = azimuth;
    orientationInDegree[1] = pitch;
    orientationInDegree[2] = roll;

    //If the acquisition phase is in course, and there is a relative movement
    clockwise
    if (isTakingPictures && (azimuth - lastAzimuth) > AZIMUTH_INCREMENT
    && (azimuth - lastAzimuth) <= 30) {
      //Frames can be acquired
      canAcquire = true;
      //The new relative orientation is updated
      lastAzimuth = (int) azimuth;
      //Update the UI
```

```
44          movementProgressBar.setProgress((int) lastAzimuth);

46          //If 360 degree completed then automatically stop the acquisition
            if (lastAzimuth >= 360) {
48            stopAndSave = true;
            }
50        }

52        //If the device position is controlled and it is in the acquisition phase and the
          user moves the phone in a not valid position
          if (controlDevicePosition && isTakingPictures && (roll <
          -INCLINATION_TRESHOLD || roll > INCLINATION_TRESHOLD || pitch <
          -INCLINATION_TRESHOLD || pitch > INCLINATION_TRESHOLD)) {
54          //Stop controlling position
            controlDevicePosition = false;
56          //Launch the Error Page
            inclinationError();
58        }
              ...
60      }
          ...
62  }
```

Listing 4.3: onSensorChanged implementation.

An important thing to be noticed is that the Android Sensor Coordinate System is different from the one used in this application, presented in the previous chapter. More precisely the Android y-axis corresponds to the z-axis and its z-axis to the -y-axis. So to work in our coordinate system a remapping of coordinates is mandatory (Listing 4.3 rows 8-9).

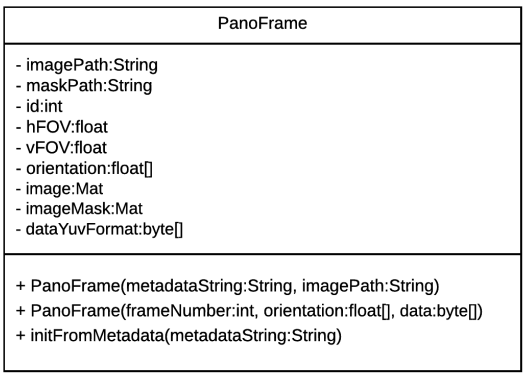


Figure 4.4: Class used to mange frames and their metadata.

Actually frames received in the *previewCallback()* are put in a queue and asynchronously saved in a background thread not to block the main thread. Frames are saved in a temporary folder to be later used in the panorama generation phase. The class *PanoFrame* is used to manage frames and their metadata, Figure 4.4.

4.3 Panorama generation

This phase begins as soon as the acquisition ends and it is executed in background. Before finishing its execution the *CameraActivity* launches the *StitchingService* class that is in charge of the execution of tasks related to the panorama generation. This class extends the Android *Service* class, a special Android component used to perform longer-running operations in background without user interactions. When launched the *StitchingService* executes its own thread in which all the work will be done. This class is the bridge between the previous module and the stitching module. In fact the second is completely independent from the rest of the application. The main class of the stitching module is the *StitchingManager*. This by means of the *composePanorama()* method enables the configuration (which algorithms and techniques to use during the process) and the launch of the stitching process. As said in chapter 3 the process is divided into two main phases: the image alignment and the actual stitching. The main steps for the generation of panoramic images are shown in Figure 4.5. Their implementation is presented in the following subsections.

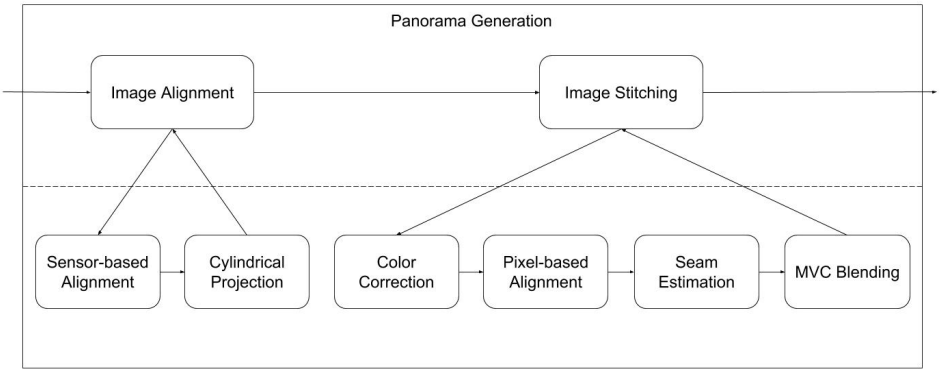


Figure 4.5: The main steps for the generation of panoramic images.

4.3.1 Image alignment

The image alignment workflow follows exactly the one presented in the previous chapter. The whole process is described in Figure 4.6. Images are sequentially read from the temporary folder where they have been saved in the acquisition phase. After being processed they are saved again in the same folder to be used later in the stitching phase. For each image a binary mask is created and saved. The binary mask identifies the useful part of an image and will be used later to select pixels during the stitching. Offsets computed in this phase correspond to the image horizontal positions on the cylindrical surface used to compose the panorama.
In figure 4.7 classes used for the image alignment implementation are shown. The *SensorBasedAligner* class manages the whole precess. It is initialized with the temporary folder path and provides the *align()* method to start the process. This method takes as input an array of *PanoFrame* containing path and metadata of each frame and returns the array of image offsets. The *PerspectiveCorrection* class provides methods used to warp images with respect to their pitch and roll values and the *CylindricalWarping* class is used to warp images on the cylindrical surface.

4.3.2 Image stitching

After the last image has been aligned the stitching phase starts. The workflow of the process is shown in Figure 4.8. The stitching process uses images produced in the previous step, image masks and the offset array to generate the final panorama. The first image is used to initialize the panoramic image and at each iteration only one image and its mask are kept in memory and stitched to the current panorama. All the steps except the blending are optional and they are executed according to some configuration parameters.
Figure 4.9 shows all the classes used for the image stitching implementation. The *SequentialStitcher* class manages the whole process. Each other class provides a specific functionality and it is instantiated basing on the initial configuration. E.g. the *LinearSpaceColorCompensator* is not used if the photo sequence has been acquired with fixed values for exposure and white balance. At the end of the composition the *SequentialStitcher* deletes all the temporary files used and returns the

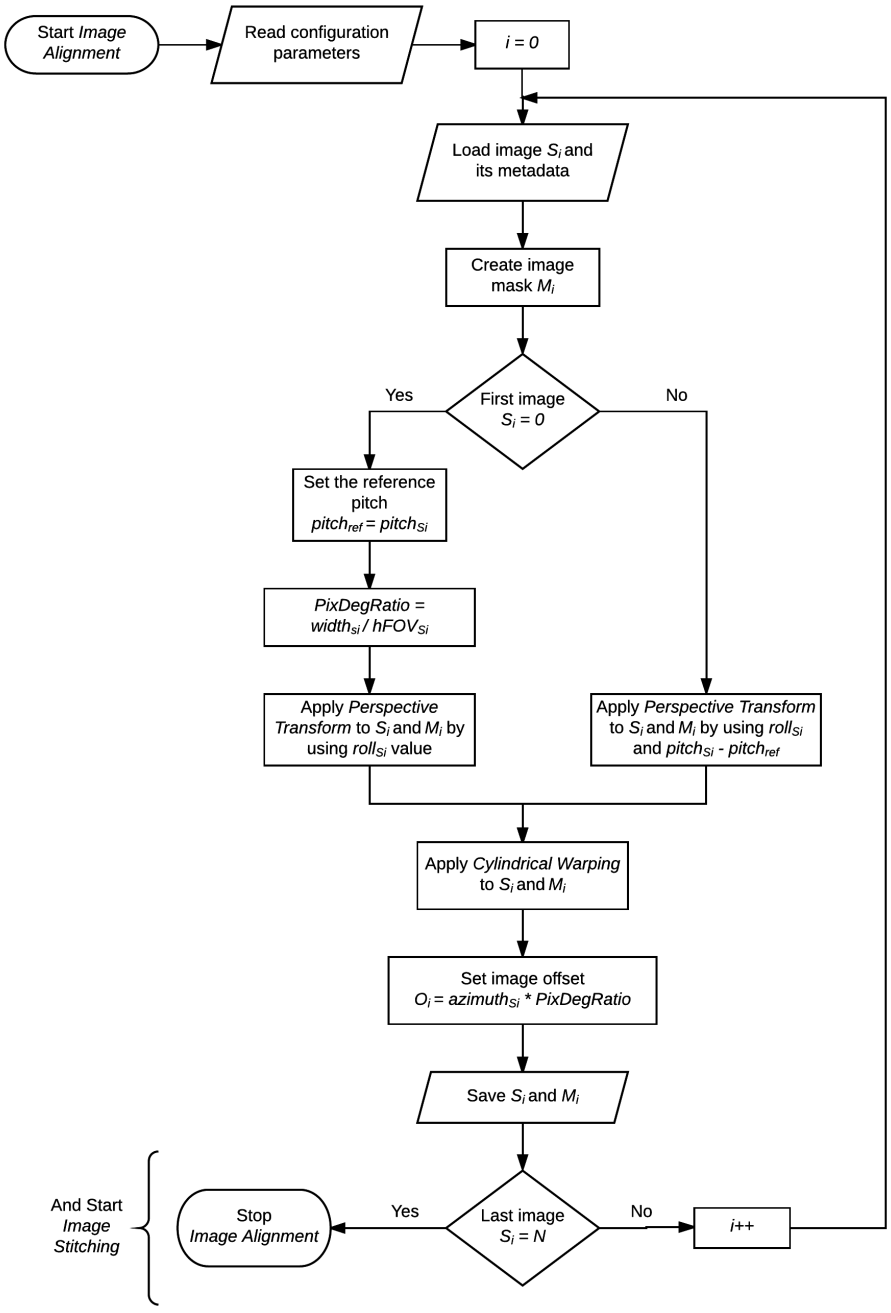


Figure 4.6: Image Alignment workflow.

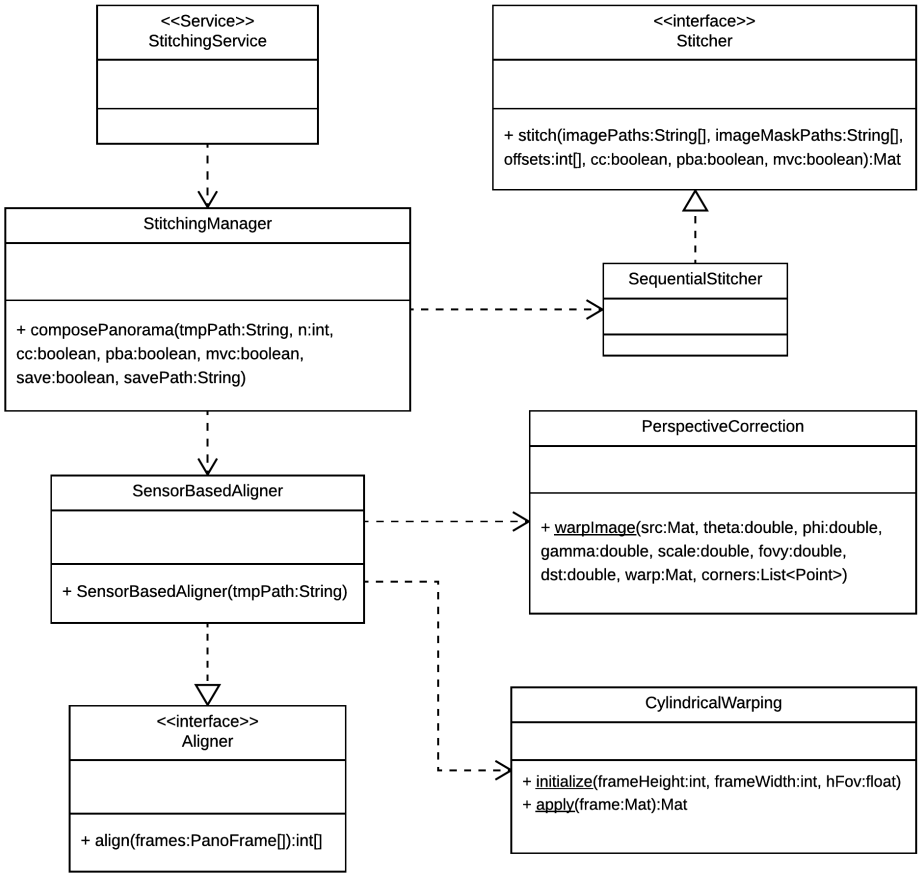


Figure 4.7: Image Alignment class diagram.

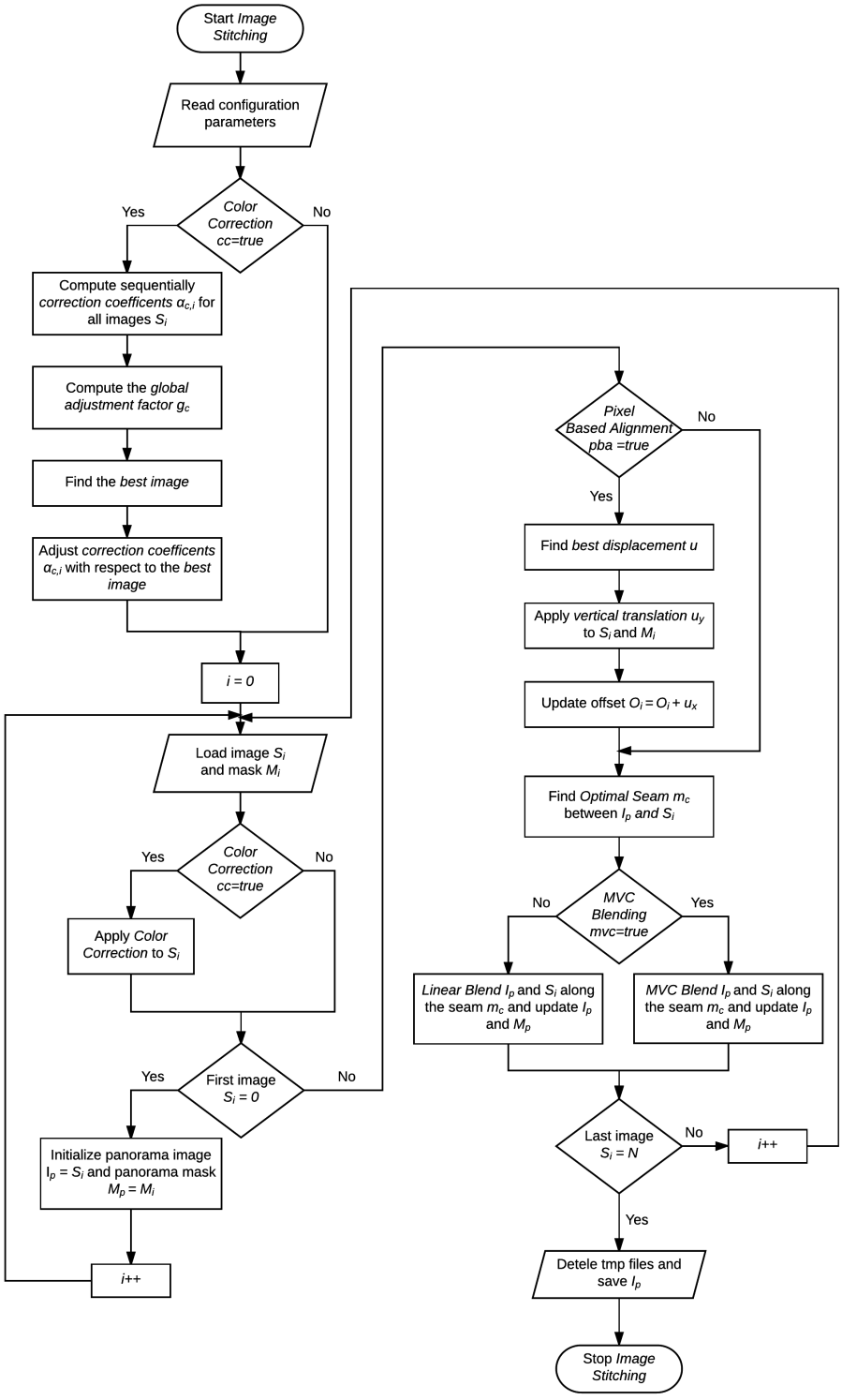


Figure 4.8: Image Stitching workflow.

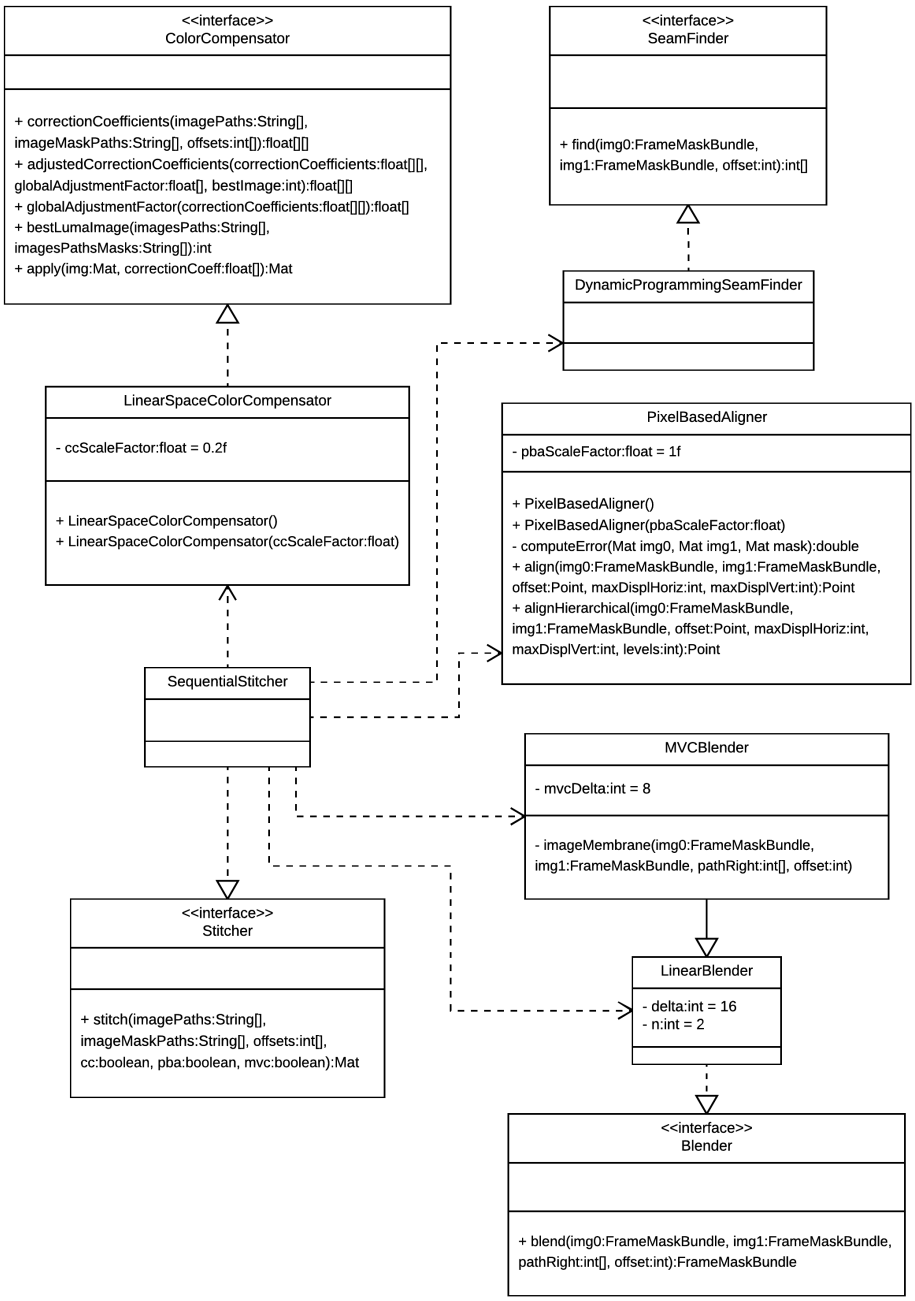


Figure 4.9: Image Stitching class diagram.

generated panorama. The *StitchingManager* computes the panorama metadata, saves the image in gallery and finishes its execution. Before finishing its execution the *StitchingService* checks if there is an Internet connection available and, in this case, it launches the peak identification service. If the connection is not available it adds the panorama to a waiting list. The peak identification service will be executed for all panoramas in the waiting list as soon as a connection is available.

4.4 Alignment and peak identification

The peak identification service has been developed in the PeakLens project. It has been imported into the application and adapted to the case of panoramic images. The service takes as input the panoramic image and some parameters such as location, orientation, fields of view and returns as output a JSON file including the list of peaks with positions and available information. This module can be configured to work in two different modes. By using just sensor data to identify peak positions or by using the global matching algorithm after the sensor estimation to refine positions. The global matching result should be more accurate than the one of sensors. An analysis and comparison of results produced in these two working modes will be done in the next chapter. An online service is used to download the representation of the skyline (which is used as reference for the global matching algorithm) as well as the list of peaks with their positions (in the Skyline Frame) and all their available information (name, height, distance). Figure 4.10 shows the main classes and methods used to implement this module.
The *PeakManager* class is the bridge between the Android application and the peak identification service. It contains all the methods for the management of peak data inside the application. There are methods to access the peak identification service, to add or remove panoramas from the waiting list and to generate a clustered peak list for the visualization phase. The *PeakFinderService*, which extends the *IntentService* class, is always used at the launch of the application to start, in a background thread, the peak identification for all the panoramas in the waiting list. The *findAndSavePeaksData()* method is used to launch the service and to save the JSON file received. This will call the *getPeaksDataFromAPI()* which initializes all the needed parameters and then invokes the *processImage()* method in the *ComputerVision* class

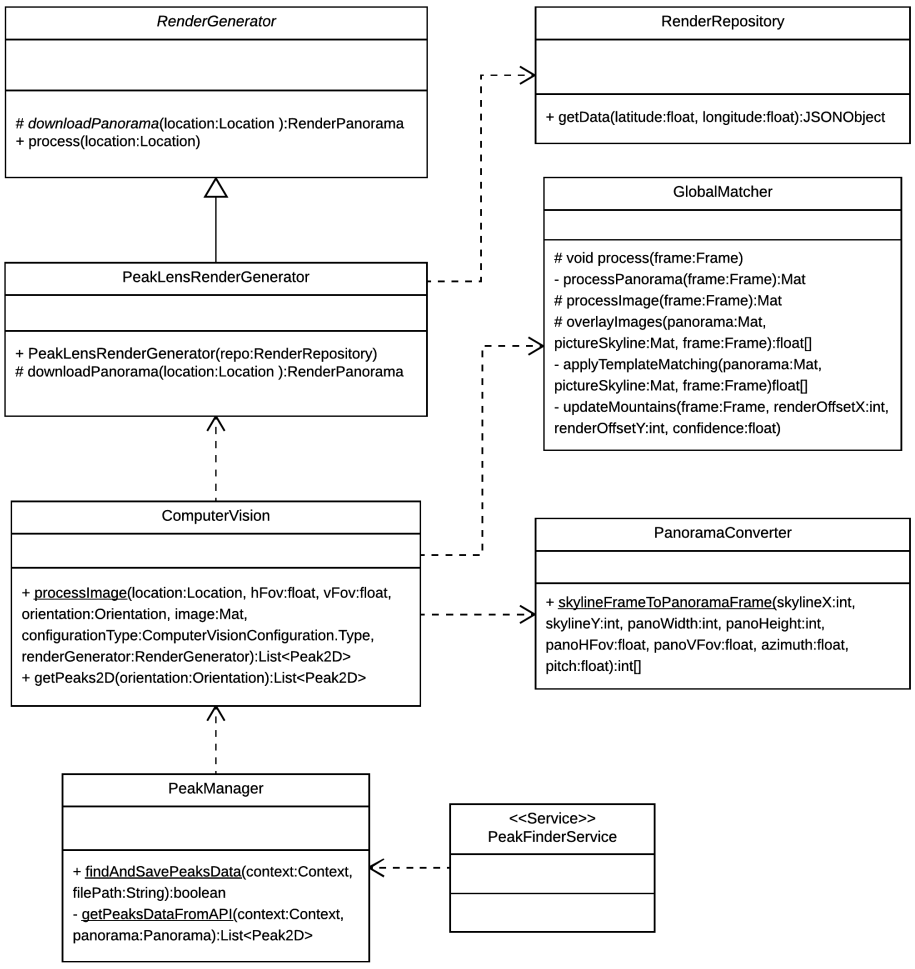


Figure 4.10: Alignment and peak identification class diagram.

to start the process. This class is the one in charge of managing all the tasks related to the identification of peaks. The *ComputerVision* class and all the classes that it uses are part of the PeakLensCV module. This is the module developed in the PeakLens project and imported and adapted to this application.

4.4.1 Sensor-based peak positioning

The *processImage()* method of the *ComputerVision* class takes as input the *configurationType* parameter. This is used to select which method to use (sensor-based or global matching). In both cases the first step is the downloading of the skyline representation and the list of peaks. The *PeakLensRenderGenerator* is in charge of the interaction with the online service to provide these data. It uses the *RenderRepository* class for the server's request: once the JSON file has been received it instantiates and returns the *RenderPanorama* class which contains the skyline image, the list of peaks and all the the peak data available for the given position. The second step is the invocation of the *getPeak2D()* method. This takes as input the orientation of the panorama and returns the list of identified peaks in the panorama coordinate system. If the *configurationType* is *ComputerVisionConfiguration.Type.SENSOR*, only a simple remapping from the Skyline Frame to the Panorama Frame will be performed. The implementation of the remapping process, which has been illustrated in the previous chapter, is shown in Listing 4.4. The remapping is performed for each peak in the list by invoking the *skylineFrameToPanoramaFrame()* method in the *PanoramaConverter* class.

```
public static int[] skylineFrameToPanoramaFrame(int skylineX, int skylineY, int
panoWidth, int panoHeight, float panoHFov, float panoVFov, float azimuth, float
pitch) {
2
        float renderPixelPerDegrees = ComputerVision.getRenderWidth() / 360;
4
        // Panorama Frame scaled with respect to Skyline Frame
6       float frameWidthScaled = panoHFov * renderPixelPerDegrees;
        float frameHeightScaled = panoVFov * renderPixelPerDegrees;
8
        // Panorama Frame positioned into the Skyline Frame
10      float azimuthDegrees = (float) ((Math.toDegrees(azimuth) + 360) % 360);
        float pitchDegrees = (float) Math.toDegrees(pitch);
12      float frameCenterX = azimuthDegrees * renderPixelPerDegrees;
```

```
        float frameCenterY = ComputerVision.getRenderOriginalHeight() / 2 +
        pitchDegrees * renderPixelPerDegrees;

        // Panorama Frame with respect to Skyline Frame
        float frameStartX = frameCenterX - frameWidthScaled / 2;
        if (frameStartX < 0)
            frameStartX = ComputerVision.getRenderWidth() + frameStartX;
        float frameStartY = frameCenterY - frameHeightScaled / 2;

        // Peak coordinate in scaled Panorama Frame
        float remappedX = skylineX - frameStartX;
        if (remappedX < 0)
            remappedX = ComputerVision.getRenderWidth() + remappedX;
        float remappedY = skylineY - frameStartY;

        // Peak coordinate in Panorama Frame
        remappedX = remappedX * panoWidth / frameWidthScaled;
        remappedY = remappedY * panoHeight / frameHeightScaled;

        // Check if the peak is inside the image
        if (remappedX >= 0 && remappedX < panoWidth && remappedY >= 0
        && remappedY < panoHeight)
            return new int[]{Math.round(remappedX), Math.round(remappedY)};

        return null;
    }
```

Listing 4.4: Peak coordinates remapping from the Skyline Frame to the Panorama Frame.

4.4.2 Global Matching and position refinement

The global matching strategy is always applied after the sensor-based positioning phase. It updates peak positions by adding an offset, which is computed by aligning the skyline returned from the online service with the skyline extracted from the panoramic image. The *GlobalMatcher* class manages this phase. The *process()* method is invoked to start the process. It receives as parameter an instance of the *Frame* class that contains the panoramic image, the reference skyline image, the panorama orientation and some other parameters needed for the computation. The first step is the adjustment of the skyline representation. The image is replicated horizontally to allow the alignment of panoramas with a very large field of view. When searching for the best position some vertical and horizontal offsets are applied. So, given that panoramas can have up to a 360 degree horizontal field of view, the

skyline representation should have at least the same FOV plus the selected offset. The *processPanorama()* method performs this task. The second step is the skyline extraction from the panoramic image. This is performed in the *processImage()* method. Once the two skylines are available, the *overlayImages()* method performs the alignment by using the *templateMatching()* method and returns the best displacement found with respect to the initial estimation. Finally the *updateMountains()* method will be invoked to update peak positions.

4.5 Visualization and sharing

Panoramas generated by using the application are saved in the public external storage so they are accessible by all apps. One can access these photos from the gallery app or from the PeakLensVR internal gallery. In the second case there are two visualization modes: the fullscreen immersive mode and the cardboard mode. In both cases a 3D scene, in which the panoramic image is rendered as texture onto a cylindrical surface, is built. The camera is placed inside the cylinder and the user can look around the scene. In the first mode some labels with peak names are placed in peak positions, if they have been detected in the previous phase. And some options are available to share the image, delete it or to relaunch the peak identification service. Each panorama is saved with its own metadata and can be shared both with other users that use the PeakLensVR app and on Facebook to be rendered in immersive mode. In Cardboard mode the scene is rendered stereoscopically. Some hotspots are rendered in peak positions as independent objects users can interact with. By looking at an object and clicking the headset button or by using a bluetooth controller it is possible to visualize a dialog with all the available information on a peak.

4.5.1 Cardboard and Fullscreen immersive mode visualization

To build the 3D environment the OpenGL ES 2.0 API has been used. It provides classes and methods for rendering 2D and 3D graphics. The Google VR SDK has been used for all the tasks related to the stereoscopic rendering and the interaction with the virtual environment in Cardboard mode. This library simplifies the most common VR

development tasks such as: lens distortion correction, head tracking, side-by-side rendering, stereo geometry configuration and user input event handling. In Figure 4.11 the main classes used to build the virtual environment, visualize it and handle the user input are shown. Actually there are many more methods than the ones reported in the figure, but later on only the ones needed to understand the rendering pipeline and user interaction are discussed.

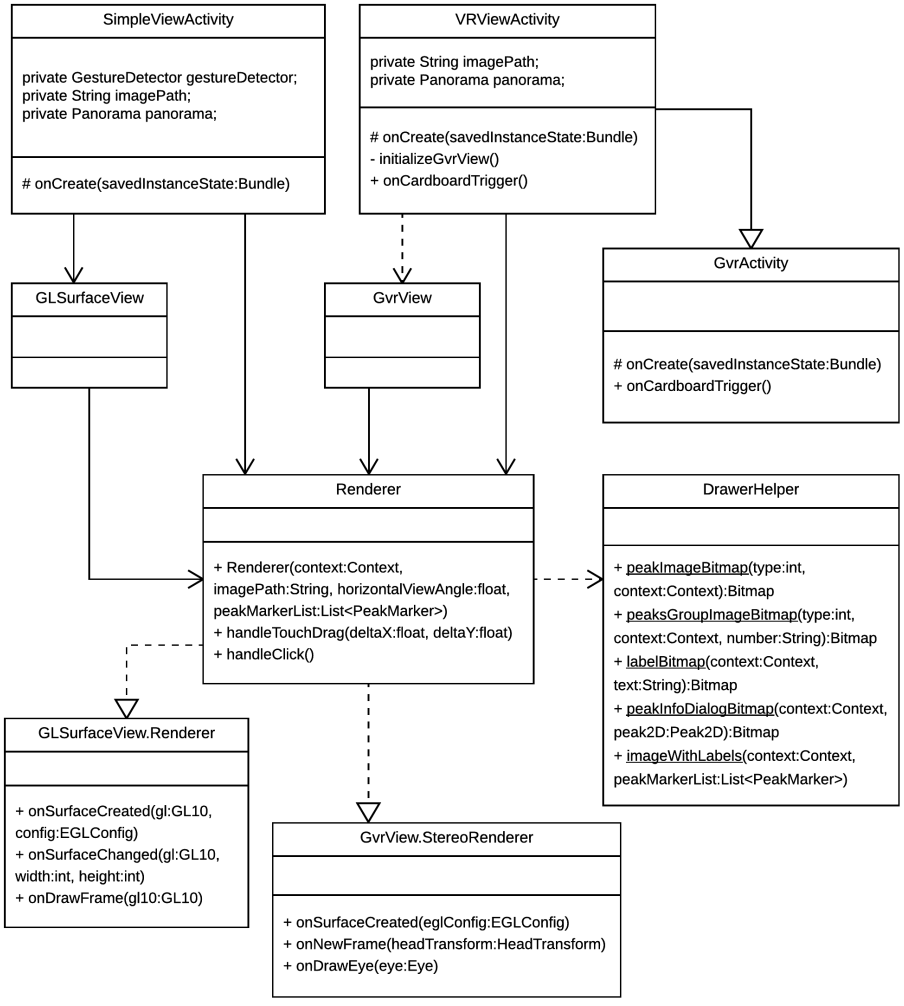


Figure 4.11: Cardboard and Fullscreen immersive mode visualization class diagram.

The *SimpleViewActivity* class deals with the visualization in fullscreen immersive mode. It has references to the *GLSurfaceView* and *Renderer*

classes. The first is a View where it is possible to draw and manipulate objects using OpenGL API calls. The second is the one used for drawing graphics on the *GLSurfaceView*. It overrides methods defined in the *GLSurfaceView.Renderer* interface. When the *SimpleViewActivity* is launched the *onCreate()* method is executed. In this method an instance of the *Render* class is created and passed to the *GLSurfaceView* class to start rendering. In its constructor the *Renderer* receives the path of the panorama, the field of view and the list of available peaks. These data will be used to initialize the scene. The last steps performed by the *SimpleViewActivity* are the instantiation of the *GestureDetector* and the registration of a *View.OnTouchListener* to detect the user interaction with the scene. All the rendering logic is implemented in the *Render* class. There are three main methods to manage the rendering pipeline:

- *onSurfaceCreated():* this method is called once, when creating the *GLSurfaceView*. It performs all the configuration and initialization operations. The graphic objects are initialized and the shaders (programs used by OpenGL for rendering) are loaded. The panoramic image with the peak labels is loaded as texture to be rendered onto the cylinder surface. To create the texture the *DrawerHelper* class is used. This class provides some utility methods to generate Bitmaps to be used as textures in the scene. The *imageWithLabels()* method receives the Bitmap with the panoramic image and the *List<PeakMarker>* that contains all the peak information and returns a Bitmap representing the image with peak labels.

- *onSurfaceChanged()*: this method is called by the system when the *GLSurfaceView* geometry changes (e.g. the screen orientation changes). Here the perspective matrix to define the viewport is initialized.

- *onDrawFrame()*: this is the core method for drawing. It is called at each new frame to draw. In this method the view matrix is modified with respect to the *xRotation* and *yRotation* values. These values reflect the scene orientation and are modified by the *handleTouchDrag()* method when the user clicks and drags on the screen to look around. Finally, the scene is drawn by using OpenGL shaders.

The *VRViewActivity* manages the Cardboard mode visualization. It extends the *GvrActivity* which is the base class to use the Google VR SDK. It provides easy integration with Google VR devices and exposes events to interact with the VR environment. It has references to the *GvrView* and *Renderer* classes. The first is provided by the Google VR SDK and is the equivalent of the *GLSurfaceView* for the *SimpleViewActivity*. The difference is that it renders content in stereo. The *Renderer* class is the same used before and is used for drawing graphics on the *GvrView*. The *Renderer* class also implements the *GvrView.StereoRenderer* interface. This delegates all stereoscopic rendering details to the view. Stereoscopic rendering and distortion correction details are abstracted from the *Renderer* and managed internally by the view. The rendering pipeline is the same described before, except that the drawing step is splitted in two phases. The new interface defines two new methods in place of the *onDrawFrame()* used before.

- *onNewFrame():* this is called at each new frame. In Cardboard mode the whole 3D environment is rendered two times, one per eye. So this method is called only once before the per-eye rendering is performed. Here all the logic to update the scene is performed. The object model matrices are updated, the check to identify selected objects is performed and the view matrix is initialized. This method receives as parameter an instance of *HeadTransform*, a class that contains the position of the head which is used to initialize the view matrix at each new frame.

- *onDrawEye():* this is called "one per-eye" and is used to perform per-eye configuration. It receives as parameter an instance of *Eye* class. The *eye* object contains the transformation and the projection matrix to be used for that eye. After the updating of these matrices the methods used for drawing are called.

The *VRViewActivity* handles the user input overriding the *onCardboardTrigger()* method. This is automatically invoked by the system when the user pulls the headset or the controller button. Each call of this method involves a call to the *handleClick()* method inside the *Render*. This method checks if a hotspot is focused or selected and updates appropriate object models.

4.5.2 Sharing

If accessed from outside the PeakLensVR application photos are simple panoramas without labels or hotspots to identify peaks. The user has to use the application gallery to access peak information. Users can share photos both from the application gallery and from the outside. In order to share the panorama with another user that wants to import it inside the application, the panorama has to be taken directly from its folder. The user that receives the panorama can import it into the application simply by placing it in the PeakLensVR folder. When the application is launched it recognizes the new image, checks for the peak JSON file and (if not present) it reads the image metadata, automatically launching the peak identification service to find peaks. Once the peak file has been downloaded the user can see the panorama with peaks. If shared by clicking the share button in the application gallery a new image with peak labels (like in fullscreen immersive mode) is generated and used for sharing. This photo will contain metadata that will be used for rendering on a platform that supports Photo Sphere XMP metadata. The Facebook 360 platform supports them and it allows one to see this kind of panoramas in immersive mode. The metadata properties that have been used are shown in Figure 4.12 and Table 4.3 taken from [16].

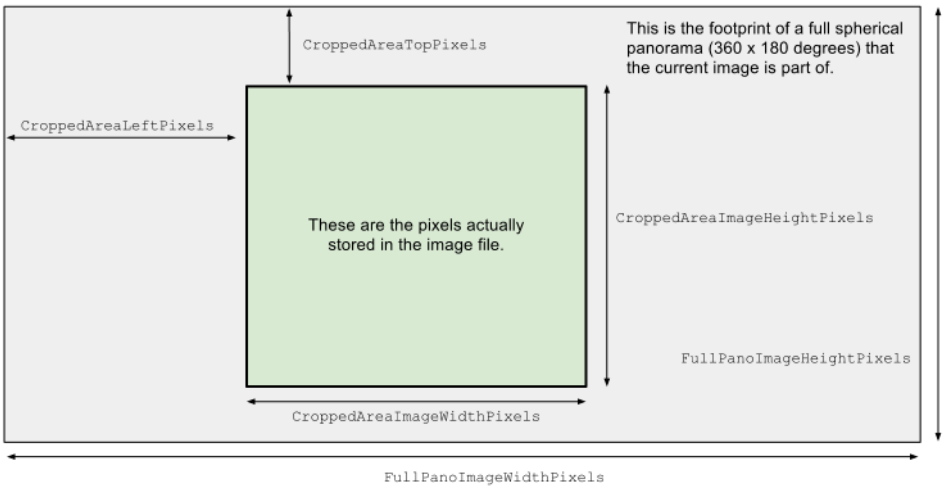


Figure 4.12: Photo Sphere XMP metadata properties.

Property	Type	Description
UsePanoramaViewer	Boolean	Photo sphere viewer or flat.
CaptureSoftware	String	App used for acquisition.
SitchingSoftware	String	App used for stitching.
ProjectionType	Text	Panorama type (e.g.equirect).
PoseHeadingDegrees	Real	Absolute azimuth in degrees.
PosePitchDegrees	Real	Absolute pitch in degrees.
PoseRollDegrees	Real	Absolute roll in degrees.
InitialViewHeadingDegrees	Integer	Initial view azimuth in degree.
InitialViewPitchDegrees	Integer	Initial view pitch in degree.
InitialViewRollDegrees	Integer	Initial view roll in degree.
FullPanoWidthPixels	Integer	Original full width.
FullPanoHeightPixels	Integer	Original full height.
CroppedAreaLeftPixels	Integer	Column from the left edge.
CroppedAreaTopPixels	Integer	Rows from the top edge.
CroppedAreaImageWidthPixels	Integer	Actual image width.
CroppedAreaImageHeightPixels	Integer	Actual image heighy.
SourcePhotosCount	Integer	Number of source photos.
ExposureLockUsed	Boolean	Auto or fixed exposure.

Table 4.3: Photo Sphere XMP metadata properties.

Chapter 5

Experimental Study

The goal of this work has been the creation of another channel for the acquisition of user-generated data suitable to environmental monitoring purpose. The innovation introduced by the project has been the application of the algorithms for the high precision peak recognition (developed through the work on the SnowWatch and PeakLens projects) to the case of panoramic pictures. All the photos generated by the application are equipped with metadata which can be used to spatially and temporally index images for the creation of environmental model and collective intelligence systems. Anyway these data, which are retrieved from smartphone sensors, are often subject to fluctuations and imprecisions. The new introduced algorithms aim at correcting these errors and identifying the image orientation with great precision . As a result the knowledge extraction process (such as the identification of peak positions) will be more accurate, so both computer systems and users will benefit from it. The effectiveness and precision of this process needs to be verified. Some tests have been performed to estimate the precision and the improvements introduced by the Global Matching algorithm with respect to sensor-based estimations. These tests have been useful not only for evaluating the Global Matching algorithm in case of very large field of view images (a case that has never been tested before) but also for evaluating sensor precision and stitching results. In fact by comparing the skyline extracted from the image with the skyline extracted from the 3D model it is possible to compute the sensor errors and to visually identify if some distortions or artifacts are introduced while generating the panorama. In this chapter the test approach will

be discussed. The data set and metrics used for the evaluation, the test execution flow and obtained results will be presented in detail.

5.1 Data set

The data set is composed by some panoramas taken in the Como lake area and in the Pollino national park between Basilicata and Calabria: some samples are shown in Figure 5.1. Photos have been taken in various weather conditions, with different exposures and, in some of them, with objects occluding the skyline. They have a horizontal field of view variable from approximately 96 degrees to 360 degrees and a number of detected peaks between 1 and 11. The data set has been produced by using a Samsung Galaxy s5 and 1280x720 pixel frames to generate panoramas. All of them have been generated by using the PeakLensVR stitching module with different configurations. Table 5.1 shows an overview of the data set.

Figure 5.1: Some samples taken from the data set.

Pano	Peaks	hFOV	Width	PixDegRatio
1	6	197,07	3734	18,95
2	11	165,45	3135	18,95
3	7	185,03	3506	18,95
4	5	360	6821	18,95
5	1	96,1	1821	18,95
6	2	357,23	6761	18,93
7	4	328,17	6217	18,94
8	1	216,33	4099	18,95
9	2	286,31	5425	18,95
10	1	184,38	3486	18,91

Table 5.1: Data set used for the evaluation.

5.2 Evaluation workflow

The goals of the tests have been:

- to measure the errors introduced by the sensor-based approach. This is useful not only to know the accuracy of the peak positions identified by using sensors but also to measure the smartphone sensor precision.
- to measure the errors introduced by the Global Matching algorithm. This is useful to verify and estimate the improvements introduced by this algorithm with respect to the sensor-based approach and the precision of the final peak positioning. In fact, both application users or computer systems, that use this service, will be aware only of final results produced by the Global Alignment algorithm.

For both approaches, errors have been computed by comparing peak positions estimated by them with the actual peak positions. The evaluation workflow is shown in Figure 5.2. The peak position ground truth has been computed manually by means of a service developed by the PeakLens team. They have built an online service that, given the geographical coordinates, returns a digital representation of the panorama visible in that point with labels for each visible peak placed in peak positions. For each photo of the dataset this service has been used to manually identify the coordinates of visible peaks. The list of

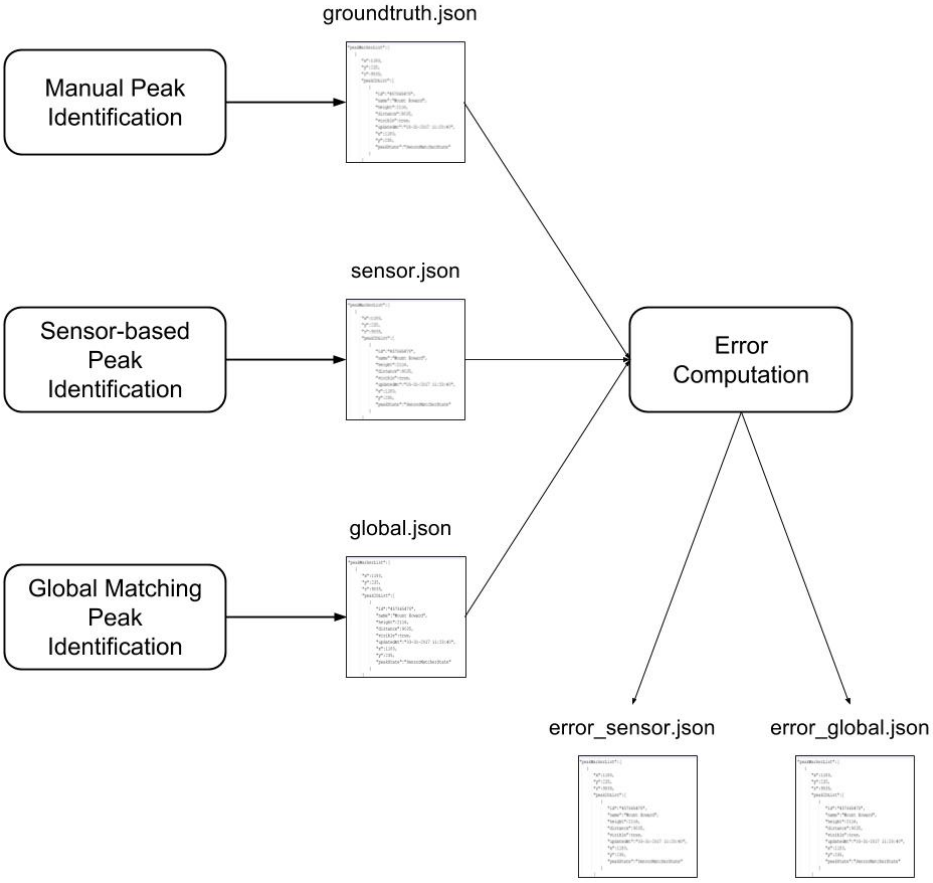


Figure 5.2: The evaluation workflow.

peak coordinates and names has been saved in a JSON file for each panorama. Then, the peak identification service has been launched for each panorama. First, the sensor-based approach has been used (PeakLensVR allows one to select the suitable approach in the setting page, Figure 5.3). The JSON files with peak positions computed by the service have been saved and then the peak identification has been relaunched by using the Global Matching approach. This will return a new JSON file for each panorama with the new position estimation. These files too have been saved. Finally for each panorama there are three JSON files: the one for the ground truth, the sensor-based estimation and the Global Matching estimation. An example of the sensor-based estimation JSON file is shown in Listing 5.1. A simple Java tool has been implemented to compute some statistics for each

Figure 5.3: Global Matching option in PeakLensVR setting page.

panorama. This tool takes as input the three JSON files and computes the error metrics that have been used for the tests. These metrics are presented in the next paragraph. Listing 5.2 shows an example of the error file returned from the tool. Results have been analyzed for each panorama and some considerations have been made starting from them and by looking at the panoramic image and the skyline generated from the digital model.

```
{
  "peakMarkerList":[
    {
      "x":1644,
      "y":505,
      "z":3001,
      "peak2DList":[
        {
          "id":"2175057377",
          "name":"Sasso di Cavallasca",
          "height":614,
          "distance":3001,
          " visible ":true,
          "updatedAt":"09-08-2017 13:26:04",
          "x":1644,
          "y":505,
          "peakState":"SensorMatcherState"
        }
      ]
    },
```

```
22          ...

24      ]
    }
```

Listing 5.1: Example of a sensor.json file.

```
1  {
       "minError": 72.69112738154499,
3      "meanError": 82.38201155240371,
       "medianError": 79.98035457910152,
5      "maxError": 96.87620966986684,
       "medianAzimuthError": 24.5,
7      "meanAzimuthError": 31.75,
       "medianPitchError": 70.5,
9      "meanPitchError": 73.25
       "errorItemList": [
11         {
               "name": "Sasso di Cavallasca",
13             "y1": 574,
               "x1": 1576,
15             "y2": 505,
               "x2": 1644,
17             "error": 96.87620966986684
               "azimuthError": 68,
19             "pitchError": 69
           },
21
           ...
23
       ]
25 }
```

Listing 5.2: Example of an error-sensor.json file.

5.3 Metrics

The metrics used for the evaluation are the Euclidean Pixel Error (EPE), the Euclidean Degree Error (EDE), the Azimuth Degree Error (ADE) and the Pitch Degree Error (PDE). The EPE is the Euclidean distance in pixels from the peak position in the ground truth and its estimated position. The EDE is the same distance measured in degrees. It has been computed by using the EPE and the *pixel-degree ratio* of each panorama visible in Table 5.1. The ADE is the distance along the

horizontal axis in degrees and the PDE the distance along the vertical axis. Let $P_1(x_1, y_1)$ be the actual peak position defined in the ground truth, $P_2(x_2, y_2)$ be the estimated position by using the sensor-based or Global Matching approach and $s = width/hFOV$ be the panorama pixel-degree ratio:

$$EPE = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \quad EDE = EPE/s$$

$$ADE = \frac{|x_1 - x_2|}{s}, \quad PDE = \frac{|y_1 - y_2|}{s}$$

All these parameters are useful to estimate the absolute errors visible in the pictures by the users as well as the sensor absolute errors. The ADE and PDE provide information on the azimuth error and the pitch error respectively. Some interesting results noticeable by observing them will be discussed in the next paragraph. Errors have been computed for each peak in each panorama and then aggregated. The min, max, mean and median EPE, the mean and median ADE and PDE have been computed for each panorama. The EDE has been considered for each peak and used to compute the frequency distribution of peaks in error classes, where classes go from 1 to 6 error degrees. There is one important thing to be considered in the evaluation of results. The peak estimation process contains an intrinsic error due to the use of the digital elevation model and an open data set for peaks. They both may contain some small imprecisions that may affect the final result. This means that the reference digital skyline may sometimes have differences from the real skyline. In this case even if the extracted skyline is perfect (it does not contain artifacts or distortions introduced in the stitching process) there could be small matching errors with the reference skyline. Anyway mountains are usually very far from the user, so these errors consist in few pixels. This problem is evident in cases in which the max error is far from the mean or median error. It could in fact be due to a peak in a position in which the reference skyline has an imprecision. The same kind of error might happen in cases of imprecisions of peak positions in the data set.

5.4 Results

Table 5.2 shows the EPE values computed for each panorama when using the sensor-based approach. These values show the fluctuations of sensor data from one panorama to another as well as the variability of errors across peaks in the same panorama. Samples 2, 4 and 6 have the largest difference between the min and max EPE. This variability may be due to some distortions introduced during the stitching process or some imprecisions in the digital elevation model and peak data set. By analyzing results after the Global Matching a more accurate hypothesis can be made. Samples 8 and 9 show the greatest sensor errors. These may be due to the use of the magnetometer. It is very sensitive to magnetic field variations, so the estimated panorama orientation might be distorted from nearby magnetic fields during the acquisition. In these cases the estimation would be completely useless and peak positions totally wrong. The mean and median values have been computed both for the mean EPE and the median EPE. In this case the median value is more significant because it is not influenced by the extreme cases of samples 8 and 9.

Pano	**Min**	**Max**	**Mean**	**Median**
1	64,88	96,02	75,21	69,8
2	12,2	122	60,46	60,41
3	27,01	53,15	42,12	42,37
4	22,2	226,04	111,7	70,71
5	86,53	86,53	86,53	86,53
6	110,22	233,36	171,79	171,79
7	72,69	96,87	82,38	78,98
8	1880,38	1880,38	1880,38	1880,38
9	1784,01	1903,09	1843,55	1843,55
10	50,6	50,6	50,6	50,6
Mean			**440,472**	**435,512**
Median			**84,455**	**74,845**

Table 5.2: Sensor-based Euclidean Pixel Errors (EPE).

Table 5.3 shows the EPE values computed for each panorama when using the Global Matching approach and the improvement with respect

Pano	Min	Max	Mean	Median	Improvement
1	14	45	32,55	34,09	56,72%
2	22,8	86,7	35,61	30,36	41,10%
3	20,59	54,64	34,79	33,54	17,40%
4	14,86	78,58	44,98	36,61	59,73%
5	41,77	41,77	41,77	41,77	51,73%
6	14,86	112,07	63,46	63,46	63,06%
7	11,4	88,56	48,46	46,95	41,18%
8	40,8	40,8	40,8	40,8	97,83%
9	50,32	78,92	64,62	64,62	96,49%
10	39,81	39,81	39,81	39,81	21,32%
Mean			**44,685**	**43,201**	**54,66%**
Median			**41,285**	**40,305**	**54,22%**

Table 5.3: Global Matching Euclidean Pixel Errors (EPE) and improvements computed on the mean with respect to the sensor-based approach.

to the sensor estimation computed on the mean EPE. By looking at results the improvements introduced by this algorithm are evident. On average errors are halved. A very interesting result is the case of Samples 8, 9. Even if their initial estimate was totally wrong, the Global Matching algorithm has been able to almost totally correct the error. There are some cases in which the improvement has not been so evident, but these are also the cases in which the initial error was smaller (Samples 3 and 10). There are some cases in which the max EPE, even if decreased, remains far from the mean and medium values. This is the case of Samples 2, 4 and 6. By analyzing the image generated superimposing the panorama and the digital skyline it is evident that, while for the greatest part of the panorama they completely match, there are some pieces in which they do not. This may be caused by the digital elevation model and peak data set imprecisions previously mentioned but in some cases it is evident that there are small errors due to the stitching process. In Sample 4 the skylines match perfectly but there are small pieces in which the digital skyline is slightly different from the image. This is due to DEM imprecisions. In Samples 6 and 7 while the centers of the skylines perfectly match, there are some differences towards the borders. These errors may be caused by the translational movement introduced during the acquisition (analyzed in chapter 3) or by some imprecisions in the field of view computation. Anyway with a

mean improvement of about 54% and a mean EPE of about 40 pixels (that corresponds to an EDE of about 2 degrees) the Global Matching results can be considered quite good.

Table 5.4 shows the mean and median ADE both for the sensor-based and Global Matching approaches and their improvements. While Table 5.5 shows their mean and median PDE. There is always a mean positive improvement for both. There are some cases in which the ADE or the PDE improvements are negative (Samples 3,7,9) but these always correspond to a preponderant positive improvement of the other value. Anyway, even if negative, the error is always less than 1 degree. It is important to notice that the ADE is always greater than the PDE. This is due to the fact that the azimuth value is computed by using the smartphone magnetometer while the pitch value is not. The magnetometer is the greatest source of imprecision for sensor data. It is in fact influenced by magnetic field variations and may produce in some cases even completely wrong measurements such as the ones of Samples 8 and 9. In these samples the strength and the importance of the Global Matching algorithm is evident. It has been able to almost totally correct the error.

	Sensor-based		**Global Matching**		
Pano	**Mean**	**Median**	**Mean**	**Median**	**Improvement**
1	1,08	1,00	1,08	1,00	0,00%
2	3,04	2,90	1,48	1,27	51,41%
3	1,09	1,21	1,30	1,16	-20,13%
4	5,43	3,27	1,89	1,11	65,18%
5	4,22	4,22	2,16	2,16	48,75%
6	9,06	9,06	3,25	3,25	64,14%
7	1,68	1,29	2,41	2,32	-44,09%
8	99,22	99,22	1,74	1,74	98,24%
9	97,29	97,29	3,17	3,17	96,75%
10	1,64	1,64	0,42	0,42	74,19%
Mean	**22,37**	**22,11**	**1,89**	**1,76**	**43,44%**
Median	**3,63**	**3,09**	**1,82**	**1,50**	**57,78%**

Table 5.4: Sensor-based and Global Matching Azimuth Degree Error ADE comparison.

After the sensor-based and Global Matching phases the EDE has been

	Sensor-based		Global Matching		
Pano	**Mean**	**Median**	**Mean**	**Median**	**Improvement**
1	3,79	3,54	1,22	1,48	67,76%
2	0,91	1,06	0,90	0,69	1,04%
3	1,85	1,95	1,21	1,11	34,71%
4	1,54	1,48	1,19	1,27	22,60%
5	1,74	1,74	0,42	0,42	75,76%
6	0,53	0,53	0,37	0,37	30,00%
7	3,87	3,72	0,66	0,58	82,94%
8	2,01	2,01	1,27	1,27	36,84%
9	0,71	0,71	1,27	1,27	-77,78%
10	2,12	2,12	2,06	2,06	2,50%
Mean	**1,91**	**1,88**	**1,06**	**1,05**	**27,64%**
Median	**1,79**	**1,85**	**1,20**	**1,19**	**32,36%**

Table 5.5: Sensor-based and Global Matching Pitch Degree Error PDE comparison.

considered for the whole set of peak measurements (and not for each panorama). These provide a measure of the final residual error perceived by users if using only the first or the second approach. Figure 5.4 shows a classification of the peak frequency distribution with respect to the EDE when only the sensor data are used. Figure 5.5 shows the same classification after the Global Matching phase. This histogram shows that the 85% of peaks has an EDE less than 3 degrees after the application of the Global Matching. This value corresponds to an EPE of about 57 pixels and can be considered a good result especially if compared with the EDE values computed for the sensor-based approach. In this case only the 35% of peaks has an EDE less than 3 degrees and the 17,5% has an EDE greater than 6 degrees.

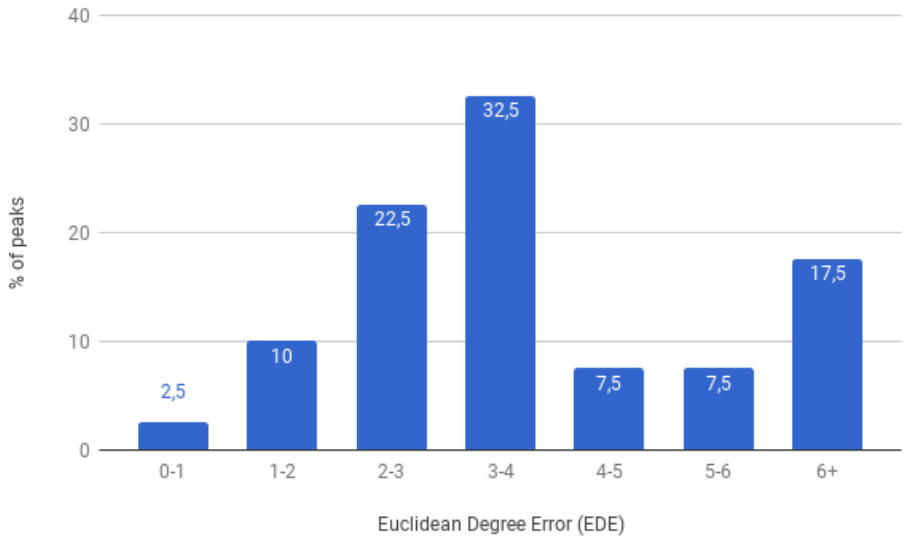


Figure 5.4: Classification of the frequency distribution of peaks with respect to their EDE after the sensor-based approach.

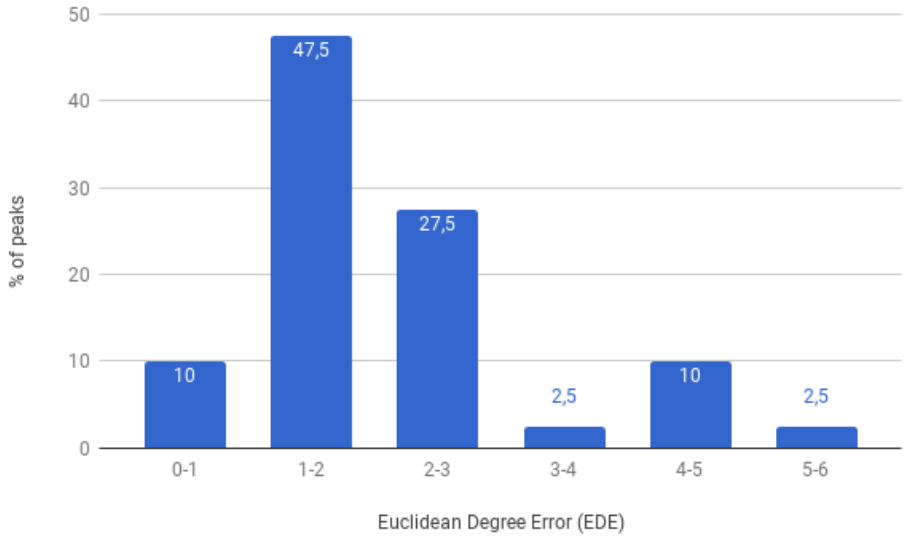


Figure 5.5: Classification of the frequency distribution of peaks with respect to their EDE after the Global Matching approach.

Chapter 6

Conclusions and Future Work

This work has led to the development of a complete mobile application that allows users to shoot panoramic photographies, automatically identifies the mountain peaks in the image and enables the visualization of photos augmented with peak information in an immersive environment. The automatic peak identification has been possible by means of the Global Matching algorithm developed through the work on the PeakLens and SnowWatch projects. The algorithm has been adapted and tested in the case of panoramic photographies with quite good results. It involves an average 54% improvement with respect to results obtained through the use of sensors. Furthermore the 85% of peak positions has an Euclidean Degree Error less than 3 degrees, as shown in chapter 5. Each application module described in this thesis could be externalized from the main application and reused in other contexts. The panorama generation module is a complete Java stitching library that can be reused in other systems that need its functionalities. The module for the alignment and peak identification can be used for example on the web to identify peaks in panoramas produced in other contexts by other applications. The module that manages the immersive visualization of panoramas and the interaction within the virtual environment can be reused to manage the creation of other virtual environments.

6.1 Future enhancements

Given its multiform nature this application could be improved in several ways. Some work could be done in improving panorama generation and peak detection techniques to provide more accurate results. The engagement factor is also quite important to stimulate the use of the application and the production of relevant amounts of data. So some work could be done in this direction by improving the user interface and global user experience.

6.1.1 Panoramic photos

The module for the generation of panoramic photos is good enough for the production of photos usable in the peak identification step. The skyline is always easily detected and it does not present relevant distortions which could make the comparison with the digital skyline difficult. Anyway in some cases, if users shoot photos with an erroneous movement, some imprecisions of the generated panorama could affect the peak positioning even in the case of a good matching between skylines. This could be avoided by introducing some techniques aimed at the compensation of the alignment errors introduced during the shooting. The introduction of more robust alignment techniques could also help the quality of panoramic photographies produced by the application. In addition to the improvement of the panorama generation module a functionality to allow users to import panoramas generated with other applications can be implemented. The Photo Sphere XMP metadata provide all the properties this application needs for the peak detection and immersive visualization stages. These metadata, that were initially introduced for the Google Street View Photo Spheres, have been recently adopted by Facebook to enable the immersive visualization of panoramas. This gives hope that all the applications that generate panoramic photos will soon adopt this standard and so all the panoramas generated by them could be imported in PeakLensVR.

6.1.2 Matching precision

In this project the Global Matching algorithm has been used. The team that has worked on PeakLens and SnowWatch projects has also developed another technique to improve the final peak positioning: the Local Matching algorithm. Once the image skyline has been aligned

with the digital skyline and the peak positions have been estimated, this algorithm performs another refinement step. For each peak a small patch of the skyline is extracted on the peak position and the alignment is performed only for that patch. An offset is computed for the best position found, so at the end each peak will have its own offset with respect to the initial position estimated through the Global Matching. This step should provide better results because it is less affected by the imprecisions of the digital skyline or of the extracted skyline. Even the problems due to alignment errors previously discussed could be solved by means of this technique. The Local Matching algorithm could be introduced and tested in future application updates.

6.1.3 User experience

The application has an internal gallery to allow users to select the panorama to be visualized. Panoramas can be viewed both in fullscreen immersive mode and in Cardboard mode. When using the Cardboard headset users must select the panorama in the gallery and then wear the headset. If they want to go back to the gallery and visualize another panorama they have to take off the headset, remove the smartphone from it, select the new image and wear the headset again. This operation could become pretty annoying while using the application. An interesting feature to be implemented could be the full virtual navigation of the application. The gallery could be accessible in the virtual environment to allow users to select and view different panoramas without having to take off the headset.

6.1.4 Crowdsourcing

To better collect user data the sharing functionalities should be improved. The user might be asked if he wants to automatically share his photos to contribute to environmental safeguard programs and an automatic sharing functionality could be implemented to share all or only selected panoramas. A mechanism based on collecting points to earn rewards can be created to stimulate the user engagement and the sharing of photos. In addition an online platform that aggregates the user-generated panoramas to automatically produce maps and virtual tours could be developed. This could lead to the birth of an online community and the production of more and more material for collec-

tive intelligence systems and environmental monitoring purposes.

Bibliography

[1] 360 photos - facebook 360 videos. https://facebook360.fb.com/360-photos/. Accessed: 2017-08-14.

[2] Adobe extensible metadata platform. http://www.adobe.com/products/xmp.html. Accessed: 2017-08-14.

[3] Android camera uml class diagram example. http://www.uml-diagrams.org/android-camera-uml-class-diagram-example.html. Accessed: 2017-08-29.

[4] Create street view in a snap. https://www.google.com/streetview/publish/. Accessed: 2017-08-17.

[5] Fotocamera cardboard. https://play.google.com/store/apps/details?id=com.google.vr.cyclops. Accessed: 2017-08-17.

[6] Google cardboard. https://vr.google.com/cardboard/. Accessed: 2017-08-14.

[7] Google street view. https://play.google.com/store/apps/details?id=com.google.android.street. Accessed: 2017-08-17.

[8] Graph cuts versus dynamic programming. http://www.cs.utexas.edu/~grauman/courses/fall2011/handouts/examples/Adrian_demo_with_notes.pdf. Accessed: 2017-08-17.

[9] Hierarchical clustering. https://en.wikipedia.org/wiki/Hierarchical_clustering. Accessed: 2017-08-29.

[10] How to calculate perspective transform for opencv from rotation angles. https://stackoverflow.com/questions/17087446. Accessed: 2017-08-29.

[11] Hugin - panorama photo stitcher. http://hugin.sourceforge.net/. Accessed: 2017-08-14.

[12] Image blending. http://graphics.cs.cmu.edu/courses/15-463/2010_spring/Lectures/blending.pdf. Accessed: 2017-08-17.

[13] Luma. https://en.wikipedia.org/wiki/Luma_(video). Accessed: 2017-08-30.

[14] Panoramic photography. https://en.wikipedia.org/wiki/Panoramic_photography. Accessed: 2017-08-14.

[15] Peaklens - mountain identification android mobile app. http://peaklens.com/. Accessed: 2017-08-14.

[16] Photo sphere xmp metadata. https://developers.google.com/streetview/spherical-metadata. Accessed: 2017-08-14.

[17] Quadtree. https://en.wikipedia.org/wiki/Quadtree. Accessed: 2017-08-30.

[18] Realtá virtuale. https://it.wikipedia.org/wiki/Realt%C3%A0_virtuale. Accessed: 2017-08-14.

[19] Ricoh theta s. https://theta360.com/en/about/theta/s.html. Accessed: 2017-08-14.

[20] Snowwatch portal. http://snowwatch.polimi.it/. Accessed: 2017-08-14.

[21] Trail me up. http://www.trailmeup.com/. Accessed: 2017-08-14.

[22] Types of panoramic photography. https://www.picturecorrect.com/tips/types-of-panoramic-photography/. Accessed: 2017-08-17.

[23] What are the practical differences when working with colors in a linear vs. a non-linear rgb space? https://stackoverflow.com/questions/12524623. Accessed: 2017-08-30.

[24] What is virtual reality. https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html. Accessed: 2017-08-14.

[25] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[26] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. Coordinates for instant image cloning. In *ACM Transactions on Graphics (TOG)*, volume 28, page 67. ACM, 2009.

[27] Roman Fedorov. Mountain peak detection in online social media. http://hdl.handle.net/10589/82521, 2012/2013.

[28] Roman Fedorov. *Exploiting public web content to enhance environmental monitoring*. PhD thesis, Politecnico di Milano, 2017.

[29] Roman Fedorov, Darian Frajberg, and Piero Fraternali. A framework for outdoor mobile augmented reality and its application to mountain peak detection. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 281–301. Springer, 2016.

[30] Colin J Ferster and Nicholas C Coops. A review of earth observation using mobile personal communication devices. *Computers & Geosciences*, 51:339–349, 2013.

[31] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[32] P. Fraternali, A. Castelletti, R. Soncini-Sessa, C. Vaca Ruiz, and A. E. Rizzoli. Putting humans in the loop: Social computing for water resources management. *Environ. Model. Softw.*, 37:68–77, November 2012.

[33] Seong Jong Ha, Hyung Il Koo, Sang Hwa Lee, Nam Ik Cho, and Soo Kyun Kim. Panorama mosaic optimization for mobile camera systems. *IEEE Transactions on Consumer Electronics*, 53(4), 2007.

[34] Seong Jong Ha, Sang Hwa Lee, Nam Ik Cho, Soo Kyun Kim, and Byungjun Son. Embedded panoramic mosaic system using auto-shot interface. *IEEE Transactions on Consumer Electronics*, 54(1), 2008.

[35] Felix Keis and Kevin Wiesner. Participatory sensing utilized by an advanced meteorological nowcasting system. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014.

[36] Edith Law and Luis Von Ahn. Input-agreement: a new mechanism for collecting data using human computation games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1197–1206. ACM, 2009.

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

[38] Christopher S Lowry and Michael N Fienen. Crowdhydrology: crowdsourcing hydrologic data and engaging citizen scientists. *GroundWater*, 51(1):151–156, 2013.

[39] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, and Luc Steels. Noisetube: Measuring and mapping noise pollution with mobile phones. *Information technologies in environmental engineering*, pages 215–228, 2009.

[40] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.

[41] Sasank Reddy, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Evaluating participation and performance in participatory sensing. *UrbanSense08*, 4, 2008.

[42] Brian L Sullivan, Christopher L Wood, Marshall J Iliff, Rick E Bonney, Daniel Fink, and Steve Kelling. ebird: A citizen-based bird observation network in the biological sciences. *Biological Conservation*, 142(10):2282–2292, 2009.

[43] Richard Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2006.

[44] Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, 3(3):177–280, 2008.

[45] Luis Von Ahn. *Human computation*. PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.

[46] Graham Whitelaw, Hague Vaughan, Brian Craig, and David Atkinson. Establishing the canadian community monitoring network. *Environmental monitoring and assessment*, 88(1):409–418, 2003.

[47] Yingen Xiong and Kari Pulli. Gradient domain image blending and implementation on mobile devices. In *International Conference on Mobile Computing, Applications, and Services*, pages 293–306. Springer, 2009.

[48] Yingen Xiong and Kari Pulli. Fast panorama stitching for high-quality panoramic images on mobile phones. *IEEE Transactions on Consumer Electronics*, 56(2), 2010.