Department of Electronic and Computer Engineering

UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

FINAL DEGREE PROJECT

June 2011

IMAGE STITCHING

Studies: Telecommunications Engineering

Author: Oscar Soler Cubero

Supervisors: Dr. Sean McGrath, Dr. Colin Flanagan

CONTENTS

ABSTRACT

Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it. Specifically, image stitching presents different stages to render two or more overlapping images into a seamless stitched image, from the detection of features to blending in a final image. In this process, Scale Invariant Feature Transform (SIFT) algorithm can be applied to perform the detection and matching control points step, due to its good properties.

The process of create an automatic and effective whole stitching process leads to analyze different methods of the stitching stages. Several commercial and online software tools are available to perform the stitching process, offering diverse options in different situations. This analysis involves the creation of a script to deal with images and project data files. Once the whole script is generated, the stitching process is able to achieve an automatic execution allowing good quality results in the final composite image.

RESUM

Processament d'imatge és qualsevol tipus de processat de senyal en aquell que l'entrada és una imatge, com una fotografia o fotograma de vídeo, i la sortida pot ser una imatge o conjunt de característiques i paràmetres relacionats amb la imatge. Moltes de les tècniques de processat d'imatge impliquen un tractament de la imatge com a senyal en dues dimensions, i per això s'apliquen tècniques estàndard de processament de senyal. Concretament, la costura o unió d'imatges presenta diferents etapes per unir dues o més imatges superposades en una imatge perfecta sense costures, des de la detecció de punts clau en les imatges fins a la seva barreja en la imatge final. En aquest procés, l'algoritme Scale Invariant Feature Transform (SIFT) pot ser aplicat per desenvolupar la fase de detecció i selecció de correspondències entre imatges a causa de les seves bones qualitats.

El desenvolupament de la creació d'un complet procés de costura automàtic i efectiu, passa per analitzar diferents mètodes de les etapes del cosit de les imatges. Diversos programari comercials i gratuïts són capaços de dur a terme el procés de costura, oferint diferents alternatives en diverses situacions. Aquesta anàlisi implica la creació d'una seqüència de commandes que treballa amb les imatges i amb arxius de dades del projecte generat. Un cop aquesta seqüència és creada, el procés de cosit d'imatges és capaç d'aconseguir una execució automàtica permetent uns resultats de qualitat en la imatge final.

RESUMEN

Procesado de imagen es cualquier tipo de procesado de señal en aquel que la entrada es una imagen, como una fotografía o fotograma de video; la salida puede ser una imagen o conjunto de características y parámetros relacionados con la imagen. Muchas de las técnicas de procesado de imagen implican un tratamiento de la imagen como señal en dos dimensiones, y para ello se aplican técnicas estándar de procesado de señal. Concretamente, la costura o unión de imágenes presenta diferentes etapas para unir dos o más imágenes superpuestas en una imagen perfecta sin costuras, desde la detección de puntos clave en las imágenes hasta su mezcla en la imagen final. En este proceso, el algoritmo Scale Invariant Feature Transform (SIFT) puede ser aplicado para desarrollar la fase de detección y selección de correspondencias entre imágenes debido a sus buenas cualidades.

El desarrollo de la creación de un completo proceso de costura automático y efectivo, pasa por analizar diferentes métodos de las etapas del cosido de las imágenes. Varios software comerciales y gratuitos son capaces de llevar a cabo el proceso de costura, ofreciendo diferentes alternativas en distintas situaciones. Este análisis implica la creación de una secuencia de comandos que trabaja con las imágenes y con archivos de datos del proyecto generado. Una vez esta secuencia es creada, el proceso de cosido de imágenes es capaz de lograr una ejecución automática permitiendo unos resultados de calidad en la imagen final.

1. INTRODUCTION

1.1. General Introduction

Signal processing is an area of electrical engineering and applied mathematics that operates and analyzes signals, in either discrete or continuous time, performing useful operations on those signals. Some of the most common signals can include sound, images, time-varying measurement values, sensor data, control system signals, telecommunication transmission signals and many others. These signals are analog or digital electrical representations of time-varying or spatial-varying physical magnitude.

It can differentiate between three types of signal processing, depending on which kind of signal is used: *analog signal processing* for not digitized signals, as radio, telephone, radar, and television systems; *discrete time signal processing* for sampled signals that are defined only at discrete points in time; and *digital signal processing* as the processing of digitised discrete time sampled signals, done by computers or specialized digital signal processors.

Digital signal processing usually aims to measure, filter and/or compress continuous real analog signals. Its first step is to convert the signal from an analog to a digital form, sampling it using an analog-to-digital converter, which converts the analog signal into a stream of numbers. However, the output signal is often another analog signal, which requires a digital-to-analog converter. Digital signal processing allows many advantages over analog processing in many applications, such as error detection and correction in transmission as well as data compression. It includes subfields like: audio and speech signal processing, sonar and radar signal processing, spectral estimation, statistical signal processing, digital image processing, signal processing for communications, and many others.

Concretely in this memory, the subcategory of signal processing analyzed is *image processing*, also usually refers to *digital image processing*. Image processing is a type of signal processing where the input is an image, such as a photograph or video frame; and the output of image processing can be an image or a set of parameters related to the image. Most image processing techniques involve treating images as a two dimensional signals and applying standard signal processing techniques. Specifically, digital image processing uses a wide range of computer algorithms to perform image processing on digital images, avoiding problems such as the increase of noise and signal distortion during the process. Medical and microscope image processing, face and feature detection, computer vision and image stitching are some of the different applications in the field of image processing.

1.2. Image

Before starting to describe the image processing and the image stitching process, it is required to understand the basic objects that it is going to work with: images. It can discriminate between still image (digital image) and moving image (digital video). In this section, first the characteristics and different formats of digital image are explained. After that, considering video stitching as the next step of image stitching, a brief introduction to the digital video is presented.

1.2.1. Still Image

In the field of engineering and computer science, it requires a kind of still image that can be manipulated by computers. For this reason it is used a numeric representation of a two-dimensional static image, known as digital image.

Firstly, to obtain a digital image from an analog image, the digitalization process is performed in some devices such as scanners or digital cameras. After that, the digital image is prepared to be processed. There are different digital image formats to work: bitmap or raster format and vector. Often, it can combine both formats in one image.

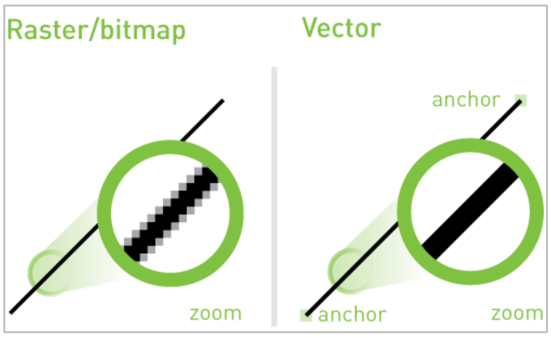


Figure 1.1. *Raster/Bitmap vs. Vector Image*

Raster Image

Raster graphic image or bitmap is composed by a serial of points, called pixels, that contains colour information. Bitmap images depend on the resolution, containing a fixed number of pixels. Each pixel has a concrete location and colour value information, what convert the pixel to the basic information unit of the image. The pixels are distributed creating a grid of cells, where each cell is a pixel, and all together build the whole image. When the grid is modified, it affects the distribution, number and colour information of every pixel, and therefore, the final image.

The image resolution is the number of pixels shown per longitude unit printed in an image, normally in pixel per inch. The quality of a bitmap image is determined in the moment of its creation, so it cannot amplify its resolution without modifying the image, normally deforming it and losing definition. This resolution is proportionally related to the file size; more pixels imply more bits.

From the previous paragraph, it is easy to see that the size is an important factor to consider in the image. The solution to manage resolution and size of images is the different compression techniques, which try to reduce the file volume with algorithms. The different raster formats achieve to reduce the image weight without modify the number of pixels. There are two types of compression: lossy and lossless techniques. The first one compresses keeping image details and colour information, while the second deletes both.

Some common bitmaps formats are:

GIF – Graphical Interchange Format: works with two methods to compress, CLUT (Colour Look-Up Table) and LZW (explained below). It is one of the most used, especially in web images, because it offers more possibilities and higher level compression than others. Suitable for 256 colours compression.

RLE – Run Length Encoding: Lossless compression technique that registers a single colour value for a group of pixels with the same colour. This technique is exploited in bitmaps images with a large amount of equal colours, saving a lot of weight and keeping the quality.

LZW – Lempel-Ziv-Welch: Similar operation to RLE.

JPEG – Join Photographic Expert Group: is one of the most adequate formats for images with more than 256 colours, appropriate for colour photos and web images compression. Although the compression can become high, the visible losses are not very significant. JPEG saves all the colour information in millions of colours without creating a large file. Discrete Cosine Transform (DCT) is the compression technique used by JPEG. It is based in divide the information in two parts, on the one hand the colour data, and on the other hand the brightness data, compressing separately too. For this reason, JPEG is not suitable for images with high contrasts of colours or for text with image. Because it is a lossy compression format, it is recommended to do the JPEG conversion in the last steps, after have done all the required modifications of the image.

PNG – Portable Network Graphics: has become important over the last times. It allows lossless compression, merging perfectly any image edge with the background. It is not able to play animated images such as GIF, and the images have more weight than in JPEG.

BMP – BitMaP: is the Windows format, very popular but its compression is poor compared with other formats such as JPEG.

PSD – PhotoShop Document: Is the format for the Adobe program, widely used because it is one of the most powerful photography programs graphically.

TIFF – Tag Image File Format: is admitted in almost all the edition and image applications. It allows many possibilities for both Mac and PC.

Vector Image

Vector images or vector graphics oriented to objects are made by vectors, objects mathematically created. The most important vector elements are Béizer curves, mathematically represented. Each vector is defined by a serial of points that have some handles to control the line shape created between them. The curve is totally defined by nodes or anchor points, and the handles. Moving the handles it can obtain the wanted curve.

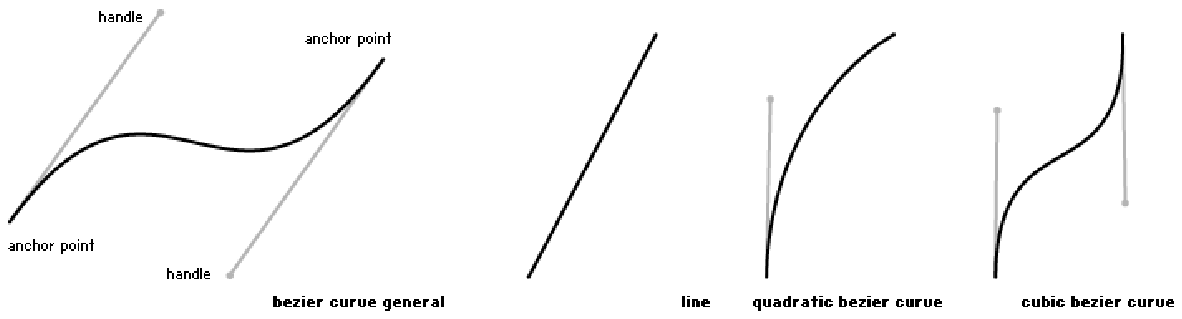


***Figure 1.2**. Different Béizer Curves.*

These lines or curves of Béizer are quite manageable because give a lot of possibilities due to their plasticity. These characteristics convert the vector images to the ideal way to work in the field of graphic design, for example in the creation of drawings or logos. The versatility of the curves makes them useful to work with text, modifying and deforming letters without limit.

Using mathematical coordinates to create images, vectorial formats allow an infinite image resolution. If an image is enlarged or reduced, its visibility will not change, nor on the screen or printed. The image conserves its forms and colours. This is the main inconvenient found in the bitmaps images.

Some of the most popular and used vector graphics formats are:

CDR – Corel DRaw: Format generated by the program with the same name.

AI – Adobe Illustrator: Characteristics similar to Corel DRaw.

EPS – Encapsulated PostScript: Very adaptable format. It is one of the best formats to be imported from most of design software.

WMF – Windows MetaFile: Format developed by Microsoft, and especially suited to work with Microsoft programs.

1.2.2. Moving Image

A moving image is typically a movie (film), or video, including digital video. Specifically, digital video is composed for a series of orthogonal bitmap digital images displayed in rapid succession at a constant rate. In the context of video these images are called frames, and typically is measured the rate at which these frames are displayed in frames per second (FPS).

There are two different formats to get the images, interlaced and progressive scan. The interlaced scan gets the image in groups of alternate lines, first the odd lines, and after the even lines, repeating progressively. In the other case, a progressive scan gets every image individually, with all scan lines being captured at the same moment in time. Thus, interlaced video captures samples the scene motion two times faster as often as progressive video does, for the same number of frames per second.

The digital video can be copied without losing quality, and many compression and encoding formats are used, such as WindowsMedia, MPEG2, MPEG4 or AVC. Probably, MPEG4 and Windows Media are widely the most used in internet, while MPEG2 is almost exclusive for DVD, giving a good quality image with minimum size.

2. IMAGE PROCESSING

Now that it has seen how image are formed, it is time to take a look at the stage of image processing, to pre-process the image and convert it into a form suitable for further analysis. This chapter reviews standard image processing operators and transforms that map pixel values from one image to another.

2.1. Point Operators

The point operators or processes are the simplest kind of image processing transforms, where each output pixel's value depends on only the corresponding input pixel value. This can be denoted as

$$g(x) = h\big(f(x)\big), \tag{2.1}$$

a function that takes one or more input images *f(x)* and produces an output image *g(x)*. For sampled images, the domain consists of a finite number of pixels locations, replacing the value $x = (i, j)$ in the equation.

Two commonly used point operators are *multiplication* and *addition* with a constant

$$g(x) = a(x)f(x) + b(x), \tag{2.2}$$

where *a* and *b* are said to control contrast and brightness, respectively.

Multiplicative gain is a linear operation related to the superposition principle.

$$h(f0 + f1) = h(f0) + h(f1). \tag{2.3}$$

Another commonly used two-input operator is the *linear blend* operator,

$$g(x) = (1 \pm \alpha)f0(x) + \alpha f1(x), \tag{2.4}$$

used to perform a temporal cross-dissolve between two images or videos.

One highly used non-linear transform applied before further processing is *gamma correction*, which is used to remove the non-linear mapping.

$$g(x) = [f(x)]^{1/\gamma}. \tag{2.5}$$

Moreover, there are the *colour transforms*, that adding the same value to each colour channel not only increases the apparent intensity of each pixel, it can also affect the pixel's hue and saturation. This colour balancing can be performed either by multiplying each channel with a different scale factor or by more complex processes.

The automatic way to determine the best values of the brightness and gain controls described before is plotting a *histogram* of the individual colour channels and luminance values. From this distribution, we can compute relevant statistics such as the minimum, maximum and average intensity values. One common solution is to perform *histogram equalization*, to find an intensity mapping function such that the resulting histogram is flat.

The trick to finding such a mapping is the same than to generate random samples from a probability density function, which is to first compute the cumulative distribution function. Integrating the distribution *h(I)* to obtain the cumulative distribution (or percentile) *c(I)*,

$$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I), \qquad (2.6)$$

it can determine the final value that pixel should take (*N* is the number of pixels in the image). When working with eight-bit pixel values, the *I* and *c* axes are rescaled from [0; 255].

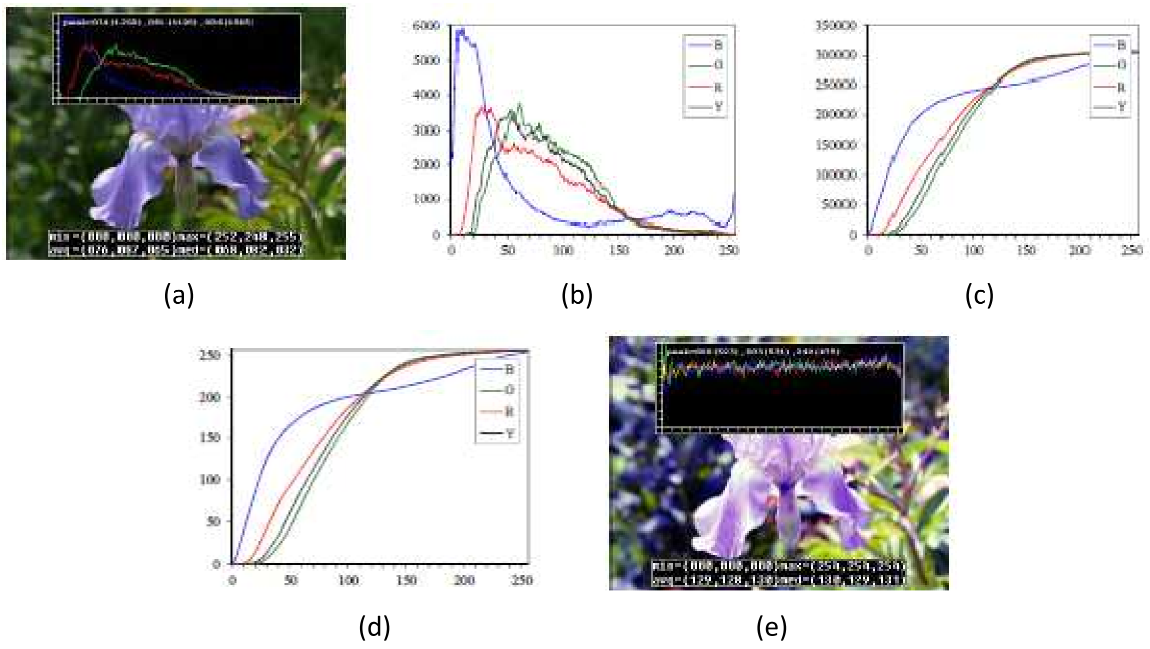


Figure 2.1. *Histogram analysis and equalization: (a) original image; (b) colour channel and intensity histograms; (c) cumulative distribution functions; (d) equalization functions; (e) full histogram equalization.*

While global histogram equalization can be useful, for some images it might be preferable different equalizations in different regions. One technique is to recompute the histogram for every *MxM* non-overlapped block centred at pixels, and then interpolate the transfer functions as it moves between blocks. This method is known as *local adaptative histogram equalization*, and is used in a variety of other applications, including the construction of SIFT (Scale Invariant Fourier Transform) feature descriptors.

2.2. Neighbourhood Operators

Locally adaptative histogram equalization is an example of neighbourhood or local operator, which uses a collection of pixel values in the surrounding area of a given pixel to determine its final output value. In addition, neighbourhood operators can be used to filter images in order to add soft blur, sharpen details, accentuate edges, or remove noise. There are linear filtering operators that involve weighted combinations of pixels in small neighbourhoods, and non-linear filtering operators such as median or bilateral filters and distance transforms.

2.2.1. Linear Filtering

The most commonly used type of neighbourhood operator is *linear filter*, in which an output pixel's value is determined as a weighted sum of input pixel values,

$$g(i,j) = \sum_{k,l} f(i+k, j+l) h(k,l). \tag{2.7}$$

The entries in the mask *h(k,l)* or kernel, are often called the *filter coefficients*. Another common variant and compactly notated formula is the *convolution* operator,

$$g = f * h\,, \tag{2.8}$$

and *h* is called the *impulse response function.* Both are *linear shift invariant* (LSI) operators, which obey both the superposition principle,

$$h\ o\ (f_0 + f_1) = h\ o\ f_0 + h\ o\ f_1 \tag{2.9}$$

and the shift invariance principle,

$$g(i,j) = f(i+k, j+l) \leftrightarrow (h\ o\ g)(i,j) = (h\ o\ f)(i+k, j+l). \tag{2.10}$$

The simplest filter to implement is the moving average or box filter, which simply averages the pixel values in a *KxK* window. It is used as a pre-processing stage to edge extraction and interest point detection algorithms.

2.2.2. Non-linear filtering

Linear filters can perform a wide variety of image transformations; however, *non-linear filters* can sometimes perform even better. One of these filters, the *median filter*, selects the median value form each pixel's neighbourhood, and is able to filter away such bad pixels. Other case is the *bilateral filter*, which simply rejects the pixels whose values differ too much from the central pixel, and the output pixel value depends on a weighted combination of neighbouring pixel values.

Other examples of neighbourhood operators include semi-global operator that computes distance transforms. The distance transform is useful in quickly precomputing the distance to a curve or set of points using a two-pass raster algorithm, and is defined as

$$D(i,j) = \min_{k,l:b(k,l)=0} d(i-k, j-l), \qquad (2.11)$$

where *d* is the distance metric between pixel offsets, and can be *Manhattan* distance or *Euclidean* distance. It has many applications, including binary image alignment, feathering in image stitching and blending, and nearest point alignment.

Finally, another practical semi-global image operation is finding *connected components*, defined as regions of adjacent pixels that have the same value or label. Connected components are used in a variety of applications, such as finding letters in scanned documents or finding objects in images.

2.3. Image Pyramids and Wavelets

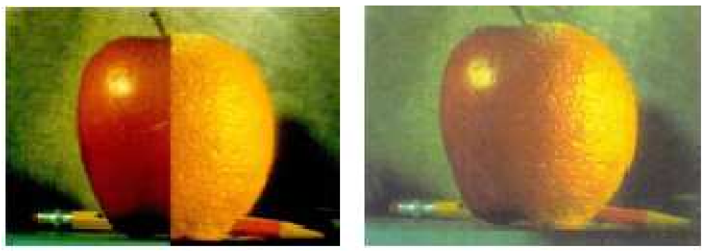
Neighbourhood operators can be cascaded to form *image pyramids* and *wavelets*, for analyzing images at a variety of resolutions and for accelerating certain operations. There are two possibilities for changing image resolution: interpolation and decimation.

In order to *interpolate* (or up sample) and image to a higher resolution, it is necessary to select some interpolation mask with which to convolve the image. On the other hand, there is *decimation* (or down sample), which is required to reduce the resolutions, where first the image convolves with a low-pass filter (to avoid aliasing) and then keep every sample.

2.3.1. Pyramids

With both techniques mentioned before, it can build a complete image pyramid, which can be used to accelerate coarse-to-fine search algorithms, to look for objects at different scales, and to perform multi-resolution blending operations. The best known and most widely used is *Laplacian pyramid*. To construct it, first the original image is blurred and subsampled by a factor two and stored in the next level of the pyramid. To compute it, first it interpolates a lower resolution image to obtain a reconstructed low-pass version from the original to yield the band-pass “Laplacian” image, stored away for further processing. The resulting pyramid has perfect reconstruction, sufficient to exactly reconstruct the original image.

One of the most engaging applications of the Laplacian pyramid is the creation of blended composite image. The approach is that low-frequency colour variations between the images are smoothly blended, while the higher-frequency textures on each one are blended more quickly to avoid ghosting effects when two textures are overlaid. This is particularly useful in image stitching and compositing applications, where the exposures may vary between different images.



(a) (b)

Figure 2.2. *Laplacian pyramid in image blending: (a) regular splice of original images (b) pyramid blend.*

2.3.2. Wavelets

An alternative to pyramids is the use of wavelet decompositions. Wavelets are filters that localize a signal in both space and frequency and are defined over a hierarchy of scales. Wavelets provide a smooth way to decompose a signal into frequency components without blocking and are closely related to pyramids.

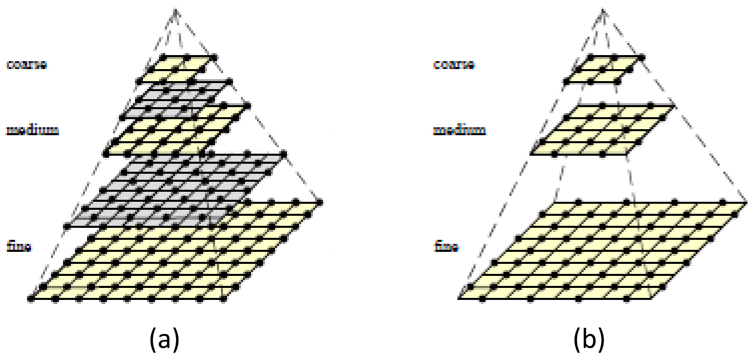


Figure 2.3. *Multiresolution pyramids: (a) pyramid with half-octave sampling; (b) wavelet pyramid, where each wavelet level stores 3/4 of the original pixels, so that the total number of wavelet coefficients and original pixels is the same.*

The main difference between pyramids and wavelets is that traditional pyramids are over complete, using more pixels than the original image to represent the decomposition, whereas wavelets keep the size of the decomposition the same as the image, providing a tight frame.

2.4. Geometric Transformations

After seeing how to change the resolution of an image in general, geometric transformations are introduced as another important class of global operators. These perform more general transformations, such as image rotations or general warps. In contrast to the point operators or processes, the functions transform the domain, $g(\boldsymbol{x}') = f(\boldsymbol{h}(\boldsymbol{x}))$, and not the range of the image. Between different geometric transformations, which most concerns to image stitching is the *parametric 2D transformation*, where the behaviour of the transformation is controlled by a small number of parameters.

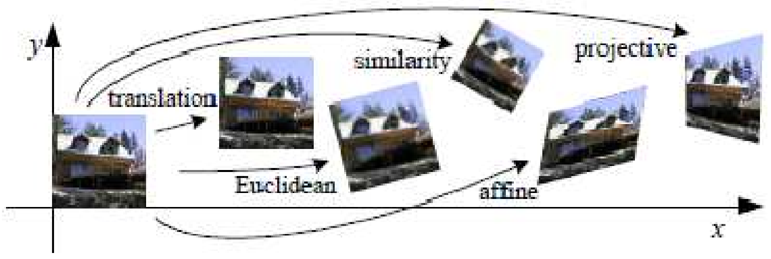


Figure 2.4. *Basic 2D geometric image transformations.*

The examples of transformations shown in *Figure 2.4.* are based on the formulas reproduced in the next table, where *I* is the inverse matrix, $R = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix}$, *s* an arbitrary scale factor, and $\tilde{H}$ and *A* arbitraries 3x3 and 3x2 matrix respectively.

Transformation	**Matrix**	**Preserves**	**Icon**
translation	$[\ \boldsymbol{I}\ \vert\ \boldsymbol{t}\ ]_{2x3}$	orientation	□
rigid (rotation+translation)	$[\ \boldsymbol{R}\ \vert\ \boldsymbol{t}\ ]_{2x3}$	lengths	◇
similarity (scaled rotation)	$[\ s\boldsymbol{R}\ \vert\ \boldsymbol{t}\ ]_{2x3}$	angles	◇
affine	$[\ \boldsymbol{A}\ ]_{2x3}$	parallelism	▱
projective	$[\tilde{\boldsymbol{H}}]_{3x3}$	straight lines	⏢

Table 2.1. *Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed.*

The process to compute the values in the new image $g(\boldsymbol{x}')$ is called *inverse warping*. Each pixel in the destination image $g(\boldsymbol{x}')$ is sampled from the original image $f(\boldsymbol{x})$. The procedure for creating the new image is the following: for every pixel $\boldsymbol{x}'$ in $g(\boldsymbol{x}')$, firstly the source location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$ is computed, and after the $f(\boldsymbol{x})$ at location $\boldsymbol{x}$ is resampled and copied to $g(\boldsymbol{x}')$. This explanation is illustrated in the next figure,

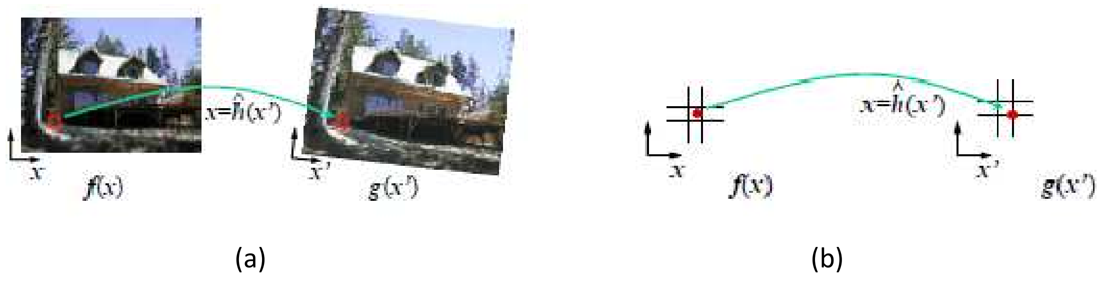


Figure 2.5. *Inverse warping algorithm: (a) a pixel sampled from its corresponding location; (b) detail of the source and destination pixel locations.*

where $\hat{\boldsymbol{h}}(\boldsymbol{x}')$ is often simply computed as the inverse of $h(\boldsymbol{x})$. Since $\hat{\boldsymbol{h}}(\boldsymbol{x}')$ is defined for all pixels in $g(\boldsymbol{x}')$, there are not holes in the result image.

3. STITCHING PROCESS

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision. Image stitching is the process of combining multiple images with overlapping fields of view to produce high-resolution photo-mosaics used for today's digital maps and satellite photos. Image stitching algorithms can create wide-angle panoramas, and they also come bundled with most digital cameras.

Since the pictures are taken until the creation of the stitched image, there are different processes to follow, starting with the detection of points or features of the single images, and ending with image merging. The image stitching processes can be classified in three main modules: registration, optimization and blending.

In this chapter, first a short history of image stitching is given as a context situation, continuing with the different stage of the stitching process, describing and covering in detail each stage.

3.1. History

Image stitching originated in the photographic community, where more manually intensive methods based on surveyed ground control points or manually registered tie points have long been used to register aerial photos into large-scale photo-mosaics. One of the key advances in this community was the development of bundle adjustment algorithms, which could simultaneously solve for the locations of all of the camera positions, thus yielding globally consistent solutions. Another recurring problem in creating photo-mosaics is the elimination of visible seams, for which a variety of techniques have been developed over the years.

In film photography, special cameras were developed in the 1990s to take ultra-wide angle panoramas, often by exposing the film through a vertical slit as the camera rotated on its axis. In the middle of 1990s, image alignment techniques started being applied to the construction of wide-angle seamless panoramas from regular hand-held cameras. More recent work in this area has addressed the need to compute globally consistent alignments to remove ghosting due to parallax error and object movement, and to deal with varying exposures. These techniques have spawned a large number of commercial stitching products.

3.2. Image Registration

Image registration involves the detection and matching pixels or features in a set of images. After that, it estimates the correct alignments relating various pairs o group of images. Before it can register and align images, it needs to establish the relationships of the pixel coordinates from one image to another, which is done with the parametric motion models shown in the previous chapter. Depending on the technique used in registration and alignment, it can consider two different methods: pixel-based method and feature-based method. Because of the development of the stitching script in chapter five, it is given more importance to the feature-based registration method, and it is explained in more detail.

3.2.1. Direct (Pixel-Based) Registration

This approach consists in to warp the images relative to each other and to look at how much the pixels agree, using pixel to pixel matching. It is often called *direct method*. To use this method, first a suitable error metric must be chosen to compare the images. After that, a suitable search technique must be devised, where the simplest technique is to do a full search. Alternatively, hierarchical and Fourier transforms can be used to accelerate the process.

The simplest way to establish an alignment between two images is to warp one image to the other. Given a template image $I_0(\boldsymbol{x})$ sampled at discrete pixel locations $\boldsymbol{x}_i = (x_i, y_i)$, the goal is to find where is located in image $I_1(\boldsymbol{x})$. A least-squares solution is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{SSD}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \qquad (3.1)$$

where $\boldsymbol{u} = (u, v)$ is the displacement.

The above error metric can be made more *robust* to outliers by replacing the squared error terms with a robust function $\rho(\,)$,

$$E_{SRD}(\boldsymbol{u}) = \sum_i \rho(I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)), \qquad (3.2)$$

that grows less quickly than the quadratic function associated with least squares. One robust $\rho(\,)$ possibility can be a smoothly varying function that is quadratic for small values but grows more slowly away from the origin. It is called *Geman-McClure* function,

$$\rho_{GM}(x) = \frac{x^2}{1+{x^2}/{a^2}}\,, \qquad (3.3)$$

where is a constant outlier threshold.

When stitching a mosaic, unwanted foreground objects have to be erased, because some of the pixels being compared may lie outside the original image boundaries. Then, the error metric become the *weighted* SSD function,

$$E_{WSSD}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u})[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{3.4}$$

and the weighting functions w_0 and w_1 are zero outside the valid range of the images.

Often, the two images to stitch were not taken with the same exposure. The *bias* and *gain* model is a simple model of linear intensity variation between the two images,

$$I_1(\boldsymbol{x} + \boldsymbol{u}) = (1 + \alpha)I_0(\boldsymbol{x}) + \beta \;\Rightarrow\; E_{BG}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - (1 + \alpha)I_0(\boldsymbol{x}_i) - \beta]^2, \tag{3.5}$$

with β and α as the bias and gain respectively. An alternative to taking intensity differences is to perform *cross-correlation* to maximize the product of the two images,

$$E_{CC}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u}). \tag{3.6}$$

To accelerate these search processes, *hierarchical motion estimation* is often used, where an image pyramid (discussed in the previous chapter) is first constructed, and a search over a smaller number of discrete pixels is first performed at coarser levels. The model estimates from one level of the pyramid, and initializes a smaller local search at the next finer level.

This estimation is not sufficient if the search range corresponds to a significant fraction of the larger image, and a *Fourier-based* approach may be preferable. Fourier-based alignment relies on the fact that Fourier transform of a warped signal has the same magnitude as the original, but linearly varying phase,

$$\mathcal{F}\{I_1(\boldsymbol{x} + \boldsymbol{u})\} = \mathcal{F}\{I_1\} e^{-2\pi j \boldsymbol{u} \cdot \boldsymbol{f}} = \mathcal{I}_1(\boldsymbol{f}) e^{-2\pi j \boldsymbol{u} \cdot \boldsymbol{f}}. \tag{3.7}$$

Consequently, to efficiently evaluate the previous models over the range of all possible values of $\boldsymbol{u}$**,** the Fourier transforms of both images are taken, and operate with both transforms, and the inverse transform is taken of the result. The Fast Fourier Transform algorithm can be significantly faster than a full search when the full range of image overlaps is considered.

Finally, to get sub-pixel precision in the alignment, incremental refinement methods based on Taylor series and parametric motion models should be used, but it is not interesting for the goals of this report to go inside the extended approximation computation.

3.2.2. Feature-Based Registration

The other possible approach to image registration is first to extract distinctive *features* from each image, to match these features establishing a global correspondence, and then to estimate the geometric transformation between the images. This kind of approach has more recently popularity for image stitching applications, and it will be used in the development of the script in the next chapter.

In this subsection, first it is explained the feature detection (extraction) stage, where each image is searched for locations that are likely to match well in other images. At the feature description stage, each region around detected keypoint locations is converted into a more compact and stable (invariant) descriptor that can be matched against other descriptors. The feature matching stage efficiently searches for likely matching candidates in other images. Finally, the last step is estimates the motion parameters that best register the images.

There are two main approaches to finding feature points and their correspondences. The first is to independently detect features in all the images under consideration and then match features based on their local appearance. The second is to find features in one image that can be accurately tracked using a local search technique, such as correlation or least squares. The first approach is more suitable when a large amount of motion or appearance change is expected, e.g., in stitching together panoramas, while the second is more suitable when images are taken from nearby viewpoints or in rapid succession. Here it is explained the first approach, due to the relevance of the panoramas creation.

Feature Detectors

The first kind of feature that you may notice are specific locations in the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized feature are often called *keypoint* features or *control points* and are often described by the appearance of patches of pixels surrounding the point location.

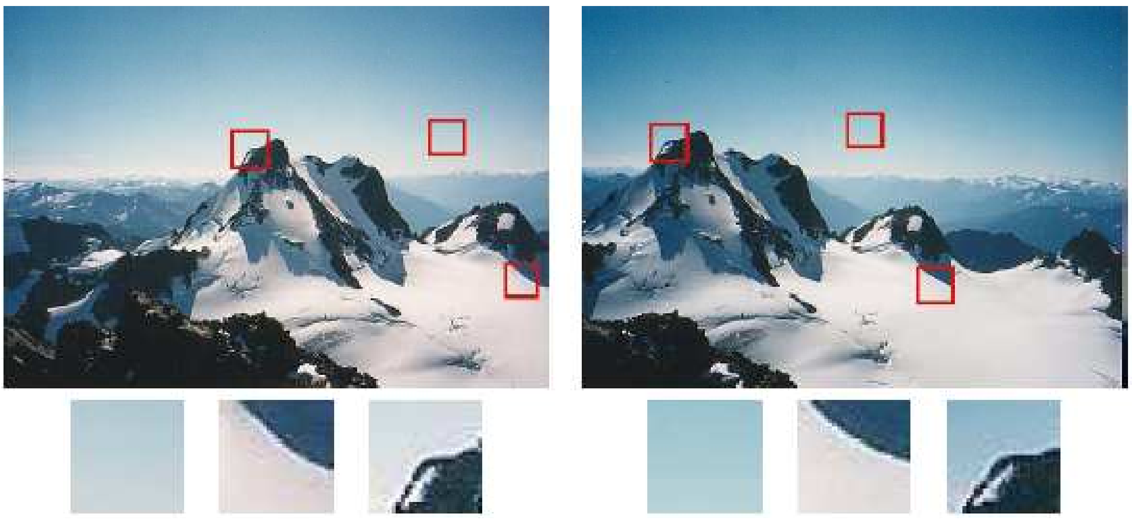


Figure 3.1. *Image pairs with three extracted patches below. Some patches can be localized with higher accuracy than others.*

Normally, texture-less patches are nearly impossible to localize, while patches with large contrast changes (or gradients) are easier to localize, although it is only possible to align patches along the direction normal to the edge direction. Patches with gradients in at least two different orientations are the easiest to localize.

These can be formalized by comparing two images patches, in a *weighted summed square difference*,

$$E_{WSSD}(\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{3.8}$$

where I_0 and I_1 are the images being compared, $\boldsymbol{u}$ the displacement vector, $w(\boldsymbol{x})$ is a spatially varying weighting function, and the summation i is over all the pixels in the patch. Moreover, if it is wanted how stable this metric is with respect to small variations in position $\Delta\boldsymbol{u}$, it is needed to compare an image patch against itself, known as an *auto-correlation* function,

$$E_{AC}(\Delta\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_1(\boldsymbol{x}_i + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \Delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}\Delta\boldsymbol{u}\ , \tag{3.9}$$

approximating with a Taylor series expansion of the image function, where $\boldsymbol{A}$ is the auto-correlation matrix, and is written as $\boldsymbol{A} = w * \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$.

Sometimes, feature detectors can lead to an uneven distribution of control points across the image. One solution is to use *adaptative non-maximal suppression* (ANMS), which only detects features whose response value is significantly greater than that of all of its neighbours within a certain radius.

In many situations, detecting features at the finest stable scale possible may not be appropriate. The way to do it is extracting features at a variety of scales, for example by performing the same operations at multiple resolutions in a pyramid and then matching features at the same level. In addition, it is required to work with in-plane image rotation, estimating a dominant orientation at each detected control point. Once its local orientation and scale have been estimated, a scaled and oriented patch around the detected keypoint can be extracted and used to form a feature descriptor. This approach is suitable when the images being matched do not suffer large scale changes and is called *multi-scale oriented patches* (MOPS).

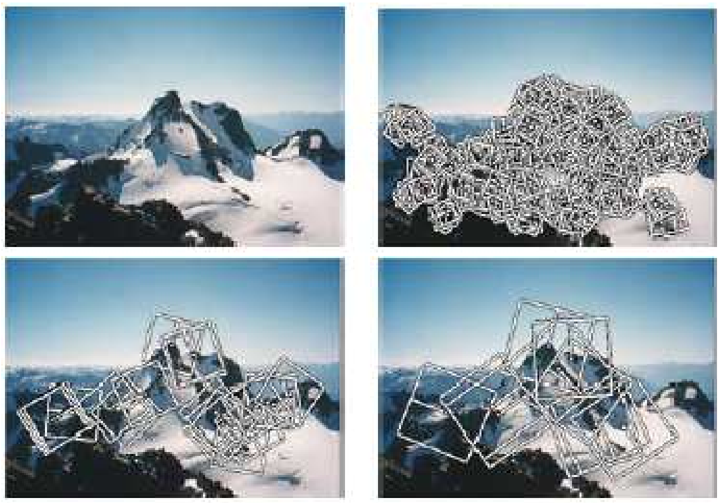


Figure 3.2. *Multi-scaled oriented patches (MOPS) extracted at three pyramid level. The boxes show feature orientation at different scales.*

Feature Descriptors

After detecting control points, in most cases, the local appearance of features changes in orientation and scale, sometimes even undergoes affine deformations, and usually varies from image to image. A *feature descriptor* is created by first computing the gradient magnitude (or patches with large contrast changes) and orientation at each image sample point in a region around the control point location. These keypoints descriptors are created to make the features points detected more invariant to such changes, still discriminating between different patches. There are a few descriptors which can be used to improve the keypoints found.

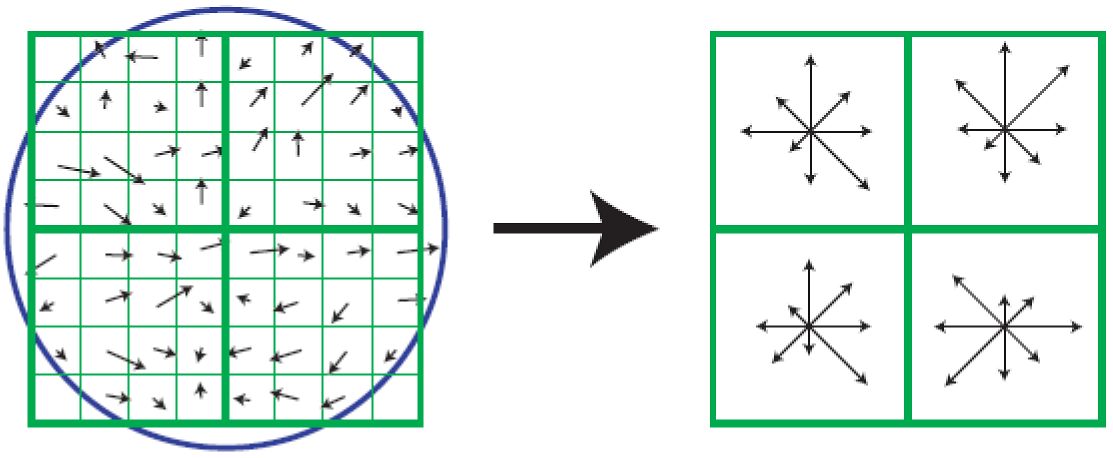


Figure 3.3. *Representation the computation of a feature descriptor*

A simple normalized intensity patches perform reasonably well and are easy to implement. The multi-scaled oriented patches (MOPS) are sampled, using a coarser level of the image pyramid to avoid aliasing. To compensate for affine photometric variations, patch intensities are re-scaled by *normalization*.

Scale invariant feature transform (SIFT) features are formed by computing the gradient at each pixel in a 16x16 window around the detected control point, using the appropriate level of the pyramid at which the keypoint was detected. This is done in order to reduce the influence of gradients far from the centre, as these are more affected by small misregistrations. This algorithm is widely explained in the chapter four of this report.

Another ways to compute descriptor are inspired by SIFT, for example *Using principal component analysis* (PCA-SIFT), or using box filters to approximate the derivatives and integrals used in SIFT, called SURF (*Speeded up robust features*). Another popular variant on SIFT is *Gradient location-orientation histogram* (GLOH), that usually has the best performance overall.

Feature Matching

Once the features and their descriptors have been extracted from two or more images, the next step is to establish some preliminary *feature matches* between these images. The first is to select a matching strategy that determines which correspondences are passed on to the next stage for further processing. The second is to devise data structures and algorithms to perform this matching as quickly as possible. This process depends on the context in which the matching is being performed. For image stitching, two images are given with a good quality overlap and most of the features in one image are likely to match the other image, although some may not match because they are occluded or their appearance has changed too much.

To begin with, the features descriptors have been designed so that Euclidean distances in features space can be directly used for ranking potential matches. With this distance metric, the simplest matching strategy is to set a threshold (maximum distance) and to return all matches from other images within this threshold. Setting the threshold too high results in too many false positives, i.e., incorrect matches being returned. But setting the threshold too low results in too many false negatives, i.e., many correct matches being missed. In the next figure, this behaviour is shown:

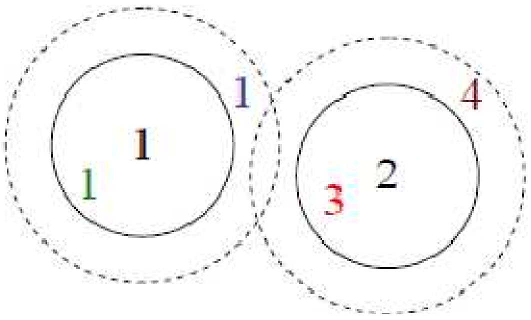


Figure 3.4. *Black digits 1 and 2 are features being matched against a database of features in other images. The solid circles are the current threshold settings. Green 1 is a true positive (good match), the blue 1 is a false negative (failure to match), and the red 3 a false positive (incorrect match). If the threshold is set higher (discontinuous circles), blue 1 becomes true positive but brown 4 becomes an additional false positive.*

It can quantify the performance of matching algorithms at a particular threshold by counting the number of true and false matches and failures, using the following definitions:

- TP (True Positives): number of correct matches.
- FN (False Negatives): matches detected incorrectly.
- FP (False Positives): proposed incorrect matches.
- TN (True Negatives): non-matches rejected correctly.

It can convert these numbers into unit rates by defining the following quantities:

- TPR (True Positive Rate), $$TPR = \frac{TP}{TP+FN} = \frac{TP}{P}; \quad (3.10)$$
- FPR (False Positive Rate), $$FPR = \frac{FP}{FP+TN} = \frac{FP}{N}; \quad (3.11)$$

with P as the number of positives.

Any particular matching strategy can be rated by TPR and FPR numbers; as the matching threshold is varying, a family of such points are obtained, which are collectively known as the *receiver operating characteristic* (ROC curve). ROC curve plots the true positive rate against the false positive rate for a particular combination of feature extraction and matching algorithms. The closer this curve lies to the upper corner (or the larger *area under the curve* (AUC)), the better its performance.

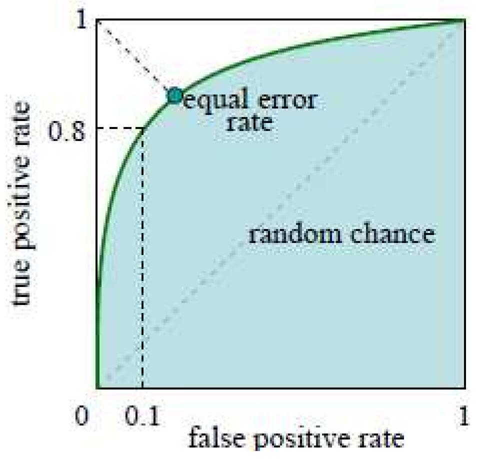


Figure 3.5. *ROC curve and its related rates. Ideally, TPR should be close to 1, while FPR is close to 0.*

Once the matching strategy is decided, it is still needed to search *efficiently* for potential candidates. The simplest way to find all corresponding feature points is to compare all features against all others in each pairs of potentially matching images. Unfortunately, this is impractical for most applications. A better approach is indexing structures as multi-dimensional search trees.

The best known of these are *kd-trees*, which divide the multi-dimensional feature space along alternating axis-aligned planes, choosing the threshold along each axis. The kd-tree recursively splits this plane along axis-aligned cutting planes. Each split can be denoted using the dimension number and split value. The *best bin first* (BBF) search searches bins in order of their spatial proximity to the query point and is therefore usually more efficient. During a BBF search, the query point first looks in its containing bin and then in its nearest adjacent bin, rather than its closest neighbour in the tree.

Feature-Based Alignment

After the feature matching stage across different images, the next step in image registration is to verify whether the set of these matching features is geometrical consistent.

Feature-based alignment is the problem of estimating the motion between two or more sets of matched points, by global parametric transformations, explained in the image processing chapter.

Given a set of matched feature points $\{(\boldsymbol{x}_i, \boldsymbol{x}'_i)\}$ and a planar parametric transformation $\boldsymbol{x}' = f(\boldsymbol{x}; \boldsymbol{p})$, it can produce a estimation of the motion parameter $\boldsymbol{p}$ using *least squares*

$$E_{pairwise-LS} = \sum_i \|\boldsymbol{r_i}\|^2 = \sum_i \|f(\boldsymbol{x_i}; \boldsymbol{p}) - \boldsymbol{x'_i}\|^2, \quad (3.12)$$

where $\boldsymbol{r_i}$ is the residual between the measured location $\boldsymbol{\hat{x}'_i}$ and its corresponding current predicted location $\boldsymbol{\tilde{x}'_i} = f(\boldsymbol{x_i}; \boldsymbol{p})$.

The above formulation assumes that all the feature points are matched with the same accuracy, and this is not often the case, because certain points may fall into more textured regions than others. With the weighted least squares function

$$E_{WLS} = \sum_i \sigma_i^{-2} \|\boldsymbol{r_i}\|^2 \;, \quad (3.13)$$

associates a scalar variance estimated σ_i^2 with each correspondence.

While regular least squares are the methods where noise follows a normal distribution, more robust versions are required when there are outliers among the correspondences. Two widely used approaches are called *RANdom Sample Consensus* (RANSAC) and *least median of squares* (LMS). Both techniques start by selecting a subset of k correspondences that is used to do an initial estimation for $\boldsymbol{p}$. Now the residuals are computed as

$$\boldsymbol{r}_i = \boldsymbol{\tilde{x}'_i}(\boldsymbol{x_i}; \boldsymbol{p}) - \boldsymbol{\hat{x}'_i} \;, \quad (3.14)$$

where $\boldsymbol{\tilde{x}'_i}$ are the estimated locations and $\boldsymbol{\hat{x}'_i}$ are the detected feature point locations.

3.3. Image Calibration

In the previous section, it has examined how to register pairs of images using both direct and feature-based methods. In most applications are given more than a single pair of images to register. In this part of *image calibration*, firstly it is required a *geometric optimization* of the alignment parameters, performing a global consistent set of parameters. Once the global alignment is computed, *local adjustments* often are needed, to minimize differences between images, such as parallax removal or blurring.

3.3.1. Bundle Adjustment

The process of simultaneously adjusting pose parameters for a large collection of overlapping images is called *bundle adjustment*, firstly applied to the general structure from motion problem and then later specialized for panoramic image stitching. The formulation of the global alignment is using a featured-based approach, because this results in a simpler system.

Considering the feature-based alignment given before in the equation 3.12, for multi-image alignment there is a collection of n features, with the location of the ith feature point in the jth image denoted by $\boldsymbol{x}_{ij}$ and its scalar inverse variance denoted by c_{ij}. Each image also has some associated pose parameters, consisting of a rotation matrix $\boldsymbol{R}_j$ and a focal length f_j. Then, to refine these estimates, it can directly extend the pairwise energy $E_{pairwise-LS}$ to a multiview formulation,

$$E_{all-pairs-2D} = \sum_i \sum_{jk} c_{ij} c_{ik} \left\| \tilde{\boldsymbol{x}}_{ik}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j, \boldsymbol{R}_k, f_k) - \hat{\boldsymbol{x}}_{ik} \right\|^2, \qquad (3.15)$$

with $\tilde{\boldsymbol{x}}_{ik} \approx \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} \tilde{\boldsymbol{x}}_{ij}$ as the predicted location of feature i in frame k, $\hat{\boldsymbol{x}}_{ij}$ is the observed location, $\boldsymbol{K}_j = diag(f_j, f_j, 1)$ as a calibration matrix, and the "2D" indicates than an image-plane error is being minimized.

An alternative way to formulate the optimization is to use true bundle adjustment

$$E_{BA-2D} = \sum_i \sum_j c_{ij} \left\| \tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j) - \hat{\boldsymbol{x}}_{ij} \right\|^2, \qquad (3.16)$$

where $\tilde{\boldsymbol{x}}_{ij} \approx \boldsymbol{K}_j \boldsymbol{R}_j \boldsymbol{x}_i$, and $\boldsymbol{x}_i$ is the point positions. It is the way to solve not only the pose estimation but also for the 3D point positions $\{\boldsymbol{x}_i\}$.

If it is wanted to minimize the error in 3D projected point directions, the previous equation changes to

$$E_{BA-3D} = \sum_i \sum_j c_{ij} \left\| \tilde{\boldsymbol{x}}_{ij}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \boldsymbol{x}_i \right\|^2, \tag{3.17}$$

and it also can derive in a pairwise energy in 3D space if the $\boldsymbol{x}_i$ points are eliminated,

$$E_{all-pairs-3D} = \sum_i \sum_{jk} c_{ij} c_{ik} \left\| \tilde{\boldsymbol{x}}_{ij}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ik}; \boldsymbol{R}_k, f_k) \right\|^2, \tag{3.18}$$

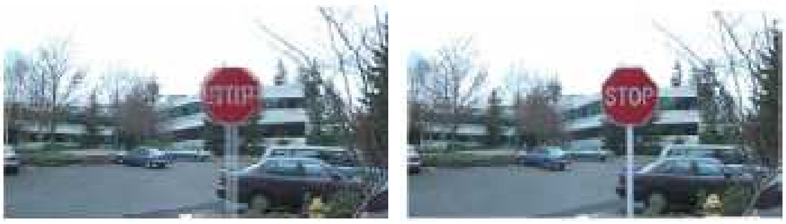
considered as global bundle adjustment formulation to optimize the poses.

3.3.2. Parallax Removal

Once optimized the global orientations and focal lengths, it may find that the results look blurry or ghosted in some places. It can be caused by a variety of factors, including radial distortion (appearance of visible curvature in the projection of straight lines), 3D parallax (rotation failure of camera around its optical centre), and small or large scale scene motions.

When the motion in a scene is very large, with objects appearing and disappearing completely, a sensible solution is to select pixels from only one image at a time as the source for the final combination. However, when the motion is reasonably small (few pixels), general 2D motion estimation can perform an appropriate correction before blending using a *local alignment*. This process is also used to compensate for radial distortion and 3D parallax.

The local alignment technique starts with the global bundle adjustment shown in equation 3.18, estimating the desired location of a 3D point $\boldsymbol{x}_i$ as the average of the back-projected 3D locations, $\tilde{\boldsymbol{x}}_i \approx \sum_j c_{ij}\, \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j)$. This can be projected into each image j to obtain a target location $\bar{\boldsymbol{x}}_{ij}$. The difference between target locations and original features $\boldsymbol{x}_{ij}$ provide a set of local motion estimates $\boldsymbol{u}_{ij} = \bar{\boldsymbol{x}}_{ij} - \boldsymbol{x}_{ij}$ which can be interpolated to form a dense correction field $\boldsymbol{u}_j(\boldsymbol{x}_j)$. Finally, the sparse $-\boldsymbol{u}_{ij}$ values are placed at the new target locations $\bar{\boldsymbol{x}}_{ij}$, then added to original pixel coordinates when computing the corrected image.



(a) (b)

Figure 3.6. *Parallax Removal. (a) Composite image with parallax; (b) image after local alignment process.*

3.4. Image Blending

After all the input images have been registered with respect to each other, it is time to decide how to produce the final stitched image. In this last stage of image stitching, *image blending* or *compositing*, involves executing the adjustments figured out in the calibration stage, combined with remapping of the images to an output projection. It also involves selecting which pixels contribute to the final composite and how to optimally blend these pixels to minimize visible seams, blur and ghosting. In this section it is explained how to choose the final composite, following by techniques for pixel/seam selection, and then the final blending process.

3.4.1. Compositing Surface

The first choice to be made is how to represent the final image. If only a few images are stitched together, a normal approach is to select one of the images as the reference and to then bend all of the other images into the reference coordinate system. The resulting composite is usually called a *flat* panorama, since the projection onto the final surface is still a perspective projection, and hence straight lines remain straight.

However, for larger fields of view it cannot maintain a flat representation without excessively stretching pixels near the border of the image. The usual choice for compositing larger panoramas is to use *cylindrical* or *spherical* projection (described in chapter 5). The choice of parameterization depends on the application, and involves a trade-off between keeping the local appearance undistorted and providing a reasonably uniform sampling of the environment.

Once it has been chosen the output parameterization, it is needed to determine which part of the scene will be centred in the final view. For a flat composite, it is normal chosen one of the images as a reference. Often, another reasonable choice is to select the image that is geometrically most central. For larger panoramas, it can still use the same method if a subset of the viewing sphere has been imaged. In the case of 360 degrees panoramas, a better option is to choose the middle image from the sequence of inputs, or sometimes the first image, considering this contains the object or image information of greatest interest.

After the parameterization and reference view have been selected, the mappings between the input and output pixel coordinates need to be computed. If the final compositing surface is flat and the input images have no radial distortion, the coordinate transformation is the simple projective transformation (shown in Table 2.1). However, if the final composite surface has some other analytic form (cylindrical or spherical), every pixel in the final panorama has to be converted into a 3D point and then map it back into each image according to the projection equations.

3.4.2. Pixel/Seam Selection and Weighting

When the source pixels have been mapped onto the final composite surface, it must still decide how to blend them in order to create an interesting stitched image or panorama. If all the images are in perfect registration, calibration and identically exposed, this is not really difficult. However, for real images, visible seams due to exposures differences, blurring due to mis-registration, or ghosting due to moving objects can occur. Creating a good panorama involves both deciding which pixels to use and how to weight or blend them.

The simplest way to create a final stitched image is to simply take an *average* value at each pixel,

$$C(\boldsymbol{x}) = \sum_k w_k(\boldsymbol{x})\tilde{I}_k(\boldsymbol{x}) \Big/ \sum_k w_k(\boldsymbol{x}) \, , \qquad (3.19)$$

where $\tilde{I}_k(\boldsymbol{x})$ are the warped images and $w_k(\boldsymbol{x})$ is 1 at valid pixels and 0 elsewhere. One way to improve it, getting better feathering is to raise the distance map values to some large power, using $w_k^p(\boldsymbol{x})$. The weighted averages then become dominated by the larger values. The resulting composite can often provide a reasonable trade-off between visible exposure and differences and blur.

In the limit $p \rightarrow \infty$, only the pixel with the maximum weight gets selected,

$$C(\boldsymbol{x}) = \sum_k \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}) \, , \qquad (3.20)$$

where $l = argmax_k \; w_k(\boldsymbol{x})$ is the label assignment or *pixel selection* function that selects which image to use at each pixel. This pixel selection process produces a visibility mask-sensitive that assigns each pixel to the nearest image centre in the set. The final composite tends to have very hard edges with noticeable seams when the exposures vary.

One way to select optimally the seams is to place these in regions where the images agree, so that transitions from one source to another are not visible. In this way, the method avoids cutting through moving objects where a seam would be not natural. For a pair of images, this technique can be a simple dynamic process starting from one edge of the overlap region and ending at the other.

If multiple images are being composite, this dynamic process is not generalized. It can observe that for well registered images, moving objects produce the most visible objects, namely *ghosts*. Then, the system has to decide which objects to keep and which ones to erase. First, all overlapping input image pairs are compared to determine *regions of difference* (RODs) where the images disagree. Next, a graph is constructed with the RODs as vertices and edges representing ROD pairs that overlap in the final composite. This technique removes regions that are near the edge of the image, which reduces the probability that partially visible objects will appear in the final composite. Once desired excess regions of difference have been removed, the final composite can be created using a feathered blend.

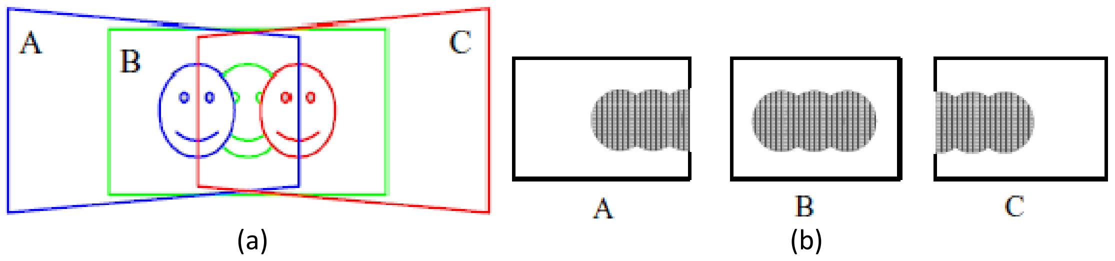


Figure 3.7. *Computation of regions of differences: (a) overlapping images whit moving object; (b) corresponding RODs caused by movement.*

3.4.3. Blending

Once the seams have been placed and unwanted object removed, the last step to blend the images is applied, in order to compensate for exposure differences and other mis-alignments. To solve these difficulties, some interesting and most used blending methods are explained below, giving different ways to merge the images into the final composite image.

Laplacian Pyramid Blending

This is a solution that uses a frequency adaptative width. First, each warped image is converted into a band-pass (Laplacian) pyramid, which involves smoothing each level with a $^1/_{16}\,(1,4,6,4,1)$ binomial mask, sub-sampling the smoothed image by a factor of 2, and subtracting the reconstructed image from the original. This creates a reversible, complete representation of the image signal, with invalid and edge pixels filled with neighbouring values to define correctly the process. Next, the mask image (with valid pixels) associated with each source image is converted to a low-pass (Gaussian) pyramid. These blurred and subsampled masks are used to perform a level feathered blend. Finally, the composite image is reconstructed by interpolating and summing all of the pyramid levels (band-pass images).

Gradient Domain Blending

An alternative method of image blending is to perform the operations in the gradient (large contrast changes) domain reconstruction. This can be used to do seamless object insertion in image editing applications. Rather than copying pixels, the gradients of the new image fragment are copied instead. The actual pixel values for the copied area are then computed by solving an equation that locally matches the gradients while following the fixed exact matching conditions at the seam boundary. After, it is also possible add a smooth function to force a stability along the seam curve.

This approach is extended to a multi-source formulation, where it no longer makes sense to talk of a destination image whose exact pixel values must be matched at the seam. In fact, each source image contributes its own gradient field, and the equation is solved using boundary conditions, concretely dropping any equations that involve pixels outside the boundary of the image.

Exposure Compensation

Pyramid and gradient domain blending are not enough when the exposure differences become large; hence this alternative approach is applied. Exposure compensation estimates iteratively a local correction between each source image and a blended composite. First, a block-based quadratic transfer function is fit between each source image and an initial feathered composite. Next, transfer functions are averaged with their neighbours to get a smoother mapping and per-pixel transfer functions are computed by interpolating between neighbouring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed and the process is repeated.

High Dynamic Range Imagining

A more principled approach is to estimate a single high dynamic range (HDR) radiance map from of the differently exposed images. It assumes that the input images were taken with fixed camera whose pixel values

$$I_k(\boldsymbol{x}) = f(c_k R(\boldsymbol{x}); \boldsymbol{p}) \tag{3.21}$$

are the result of applying radiometric transfer function $f(R, \boldsymbol{p})$ to scaled radiance values $c_k R(\boldsymbol{x})$. The form of this parametric function differs depending on the application or the goal of the stitching process. The exposures values c_k are either known by experimental setup or from a camera Exif data, or are computed as part of the proper process.

To blend the estimated noisy radiance values into a final composite, different techniques are used: a hat function (accentuating mid-tone pixels), the derivative of the response function, and optimizing the signal-to-noise ratio (SNR) that accentuates both higher pixel values and larger contrast changes in the transfer function.

Once a radiance map has been calculated, it is usually necessary to display it on 8-bit screen or printer. A variety of tone mapping techniques have been developed for this purpose, which involve either computing spatially varying transfer functions or reducing image large contrast changes to fit the available dynamic range.

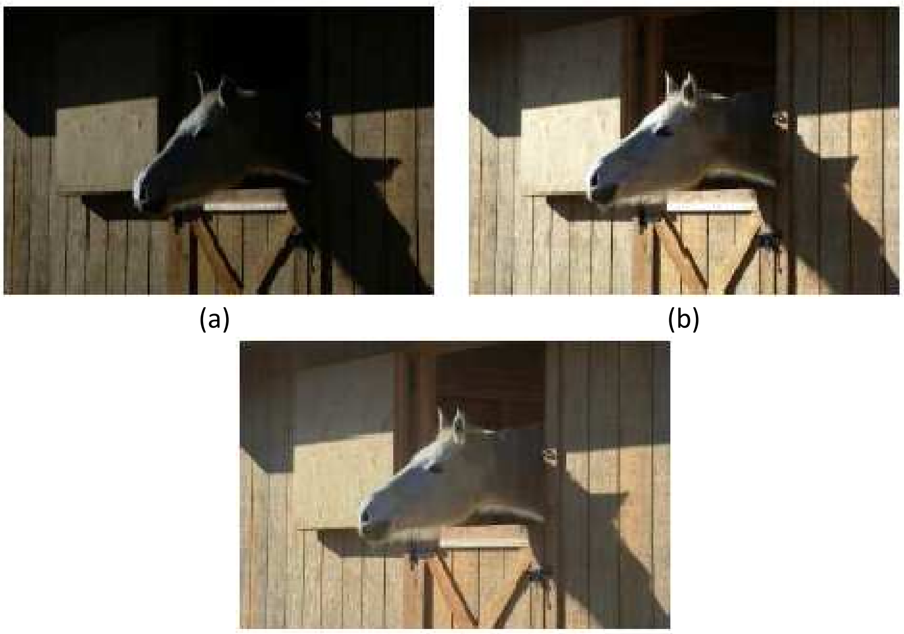


Figure 3.8. *Blending different exposures to create a high dynamic range composite: (a) and (b) two different image exposures; (c) merging the exposures in a final composite.*

4. STICHING ISSUES

The creation and execution of the stitching process cause several issues, such as the algorithms used in the registration stage, kind of problems that appear during or after the stitching, and the different software available to implement the whole process. In this chapter, first it is described some of these algorithms, next the most common problems in image stitching are presented, and finally some interesting stitching software are reviewed.

4.1. Algorithms

In this section, it is explained with more detail some of those algorithms involved in the process stitching, which are used by the tools that create the script application on this project, shown in the next chapter. These algorithms have been mentioned before in the image registration stage, and they are behind the processing of some specific tools employed for the stitching application: SIFT, RANSAC and kd-tree nearest neighbour search, which are specifically related to the control point detection and matching stage.

4.1.1. Scale Invariant Feature Transform (SIFT)

SIFT is an algorithm to detect and describe local features in images, published by David Lowe. In practice SIFT detects and uses a large number of features from the images, which reduces the contribution of the errors caused by local variations in the average error of all feature matching errors. The method can robustly identify objects even among clutter and under partial occlusion, because the SIFT feature descriptor is invariant to scale, orientation and affine distortion, and partially invariant to illumination changes.

Detection

The first stage in the algorithm is the *scale-space extrema detection*, where the interest points, which are called keypoints in the SIFT framework, are detected. The image is first convolved with $G(x, y, k\sigma)$ Gaussian-blurs images (the result of blurring an image by a Gaussian function) at different scales,

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \tag{4.1}$$

where

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.2}$$

is at scale $k\sigma$, and $I(x,y)$ is the original image. The convolved images are grouped by octave, and the value of k_i is selected, obtaining a fixed number of convolve images per octave. Then the *Difference-of-Gaussian* (DoG) images are taken from adjacent Gaussian-blurred images per octave,

$$D(x,y,\sigma) = L(x,y,k_i\sigma) - L(x,y,k_j\sigma). \qquad (4.3)$$

Now, the keypoints are identified as local minima/maxima of the DoG images across scales, comparing each pixel to its eight neighbours at the same scales and nine corresponding neighbouring pixels in each of the neighbouring scales. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint.

Localization

The next step after the production of many keypoints candidates is to perform a detailed fit of the nearby data for accurate location and scale. This information allows points to be rejected that have low contrast or are localized along an edge.

First, for each candidate keypoint, interpolation of nearby data is used to accurately determine its position. It calculates the interpolated location of the extreme using the quadratic Taylor expansion of DoG, with the candidate keypoint as the origin

$$D(\boldsymbol{x}) = D + \frac{\partial D^T}{\partial x}\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T\frac{\partial^2 D}{\partial \boldsymbol{x}^2}\boldsymbol{x}\ , \qquad (4.4)$$

evaluated at the candidate keypoint and $\boldsymbol{x} = (x,y,\sigma)$ is the offset from this point. The location of the extreme $\hat{\boldsymbol{x}}$, is determined by taking the derivative of this function with respect $\boldsymbol{x}$ and setting it to zero. If the offset is larger than 0.5 in any dimension, the extreme lies closer to another keypoint. Otherwise the offset is added to its candidate keypoint to get the interpolated estimate for the location of the extreme.

To discard keypoints with low contrast, the second-order Taylor expansion value of $D(\boldsymbol{x})$ is computed at the offset. If this value is less than 0.03, the candidate keypoint is discarded. Otherwise it is kept, with final location $\boldsymbol{y} + \hat{\boldsymbol{x}}$ and scale σ, where $\boldsymbol{y}$ is the original location of the keypoint at scale σ.

Sometimes, the DoG function has strong responses along the edges, even if the candidate keypoint is not robust to small amounts of noise. Therefore, in order to increase stability it needs to eliminate the keypoints that have poorly determined locations but have high edge responses.

Orientation

In this stage, each keypoint is assigned one or more orientations, achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and after achieve invariance to image rotation.

For an image sample $L(x,y)$ at scale σ, the gradient magnitude $m(x,y)$ and orientation $\theta(x,y)$ are calculated using pixel differences

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}, \qquad (4.5)$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}\right). \qquad (4.6)$$

These calculations are done for every pixel in a region around the keypoint. After that, an orientation histogram is formed, whose peaks correspond to dominant orientations. Finally, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peak are assigned to the keypoint.

Descriptor

Once it is ensured invariance to image location, scale and rotation, a descriptor vector is computed for each keypoint such that the descriptor is highly distinctive and partially invariant to the remaining variations such as illumination or viewpoint, and is performed on the image closest in scale to the keypoint’s scale.

A set of orientation histograms are created on 4x4 pixel neighbourhoods with 8 bins each, and computed from magnitude and orientation values of samples in a 16x16 region around the keypoint such that each one contains samples from a 4x4 sub-region of the original neighbourhood region. The magnitudes are weighted, and the descriptor becomes a vector of all the values of these histograms, it means 128 elements. Then the vector descriptor is normalized to unit length in order to enhance invariance to affine changes in illumination, and prepared for further processing.

4.1.2. Random Sample Consensus (RANSAC)

RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers, published by Robert B. Fisher. Normally, it is used at the feature matching stage, to determine good and bad keypoints connections between input images. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed.

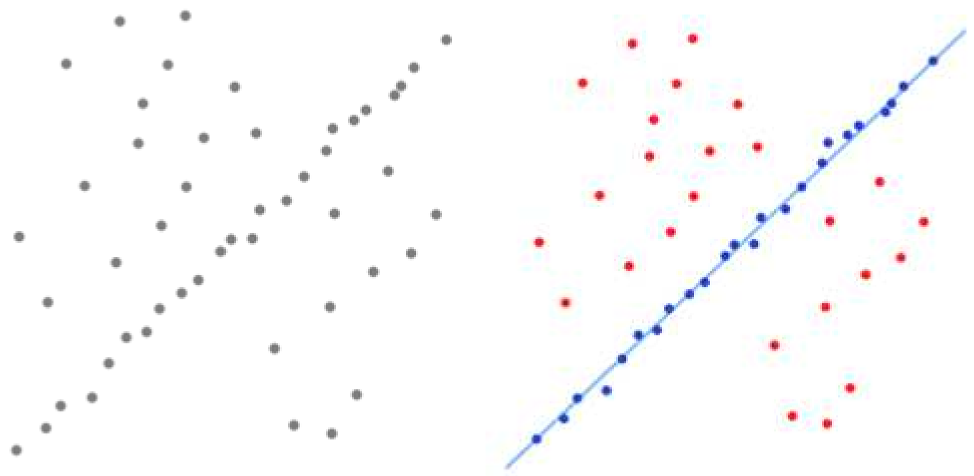


Figure 4.1. *RANSAC algorithm: (a) Data set with many outliers which line has to be fitted; (b) Fitted line with RANSAC, outliers have no influence on the result.*

The data consists of inliers of the predicted locations and outliers that do not fit the model. Usually, the outliers come from extreme values of noise or from erroneous measurements. RANSAC is the algorithm for robust fitting of models in the presence of many data outliers.

First, it is assumed that:

- Parameters $\vec{x}$ can be estimated from *N* data items.
- There are *M* data items in total.
- The probability of a randomly selected data item being part of a good model is p_g.
- The probability that the algorithm will exit without finding a good fit if one exists is p_{fail} defined as $p_{fail} = (1 - p_g^N)^L$.

Then, the algorithm is the following:

1. Selects *N* data items at random

2. Estimates parameter $\vec{x}$.
3. Finds how many data items of *M* fit the model with parameter vector $\vec{x}$ within a user given tolerance. Call this *K*.
4. If *K* is big enough, accept fit and exit with success.
5. Repeat 1...4 *L* times, defined as $= \frac{\log(p_{fail})}{\log(1-p_g^N)}$.
6. Fail if it gets here.

The refined model is kept if its error is lower than the last saved model. The good keypoint matches remain to continue the process.

4.1.3. Kd-Tree NN Search

A *k-dimensional tree* is a space-partitioning data structure for organizing points in a k-dimensional space, useful data structure for nearest neighbour searches. In feature matching, the *nearest neighbour* (NN) search is the way to find all corresponding feature points between input images. The algorithm searches keypoints in the constructed tree which are nearest to the given input keypoint.

Searching for a nearest neighbour in a kd-tree proceeds as follows:

1. The algorithm moves from the root node down the tree recursively, going right or left depending on whether the keypoint is greater than or less than the current node in the split dimension.
2. Once a leaf node is reached, that node point is saved as the current best.
3. The same process at each node: if the current node is closer than the current best, then it becomes the current best. The algorithm checks if any keypoints on the other side of the plane are closer to the search keypoint than the current best:
    a. If there are nearer keypoints on the other side of the plane, the algorithm moves down the other branch of the tree from the current node looking for closer points, following the same recursive process as step 1.
    b. If there are not, the algorithm continues going through the tree, and the branch of the other side is eliminated.

When the process is finished for the root node, the search is complete and the corresponding feature points have been matched, discarding all the incorrect matches previously detected. In this point of the stitching registration, the images are ready to be aligned, optimized and blended, following the steps commented in the previous chapter.

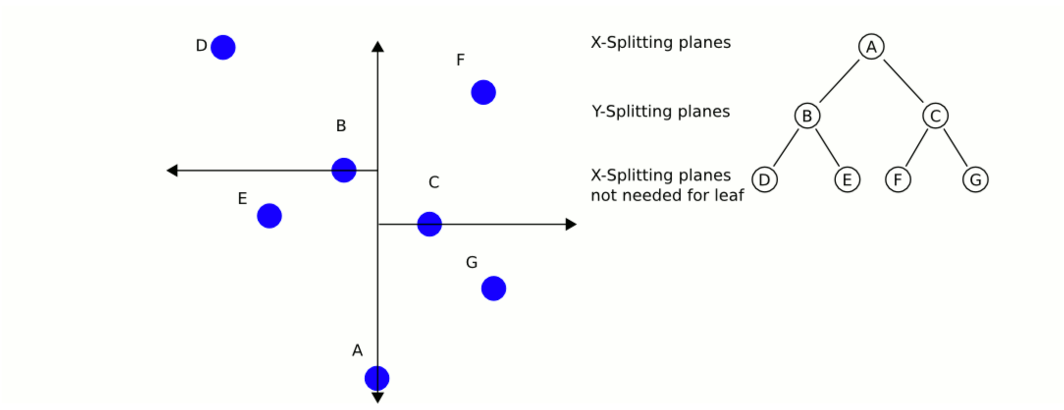


Figure 4.2. *2D Kd-tree NN search algorithm. Each interior node ('b' and 'c') represents an axis-aligned cutting plane and each leaf node is a keypoint, while node 'a' is the root node.*

4.2. Common Problems

In this section, after the stitching process has been explained, it has shown some of the most common problems found in image stitching, also in the creation of panoramas. In each issue, it is pretended to show the cause of the problem, to present some solution and give visual examples.

4.2.1. Exposure

Sometimes after stitching the input images together, it might notice some strange colours or light and dark vertical bands where the photos have been joined. These effects were not in the original photos; only appear in the finished result.

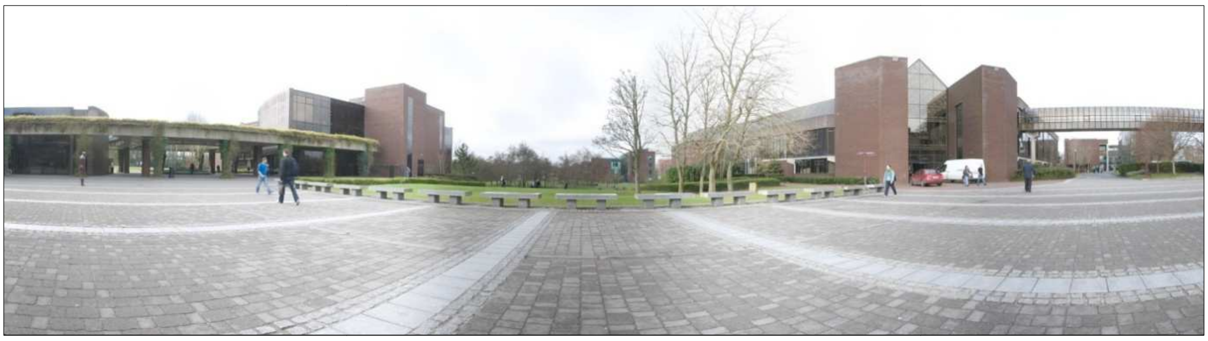


Figure 4.3. *Stitched Image with variations in exposures between frames, clearly noted in the centre of the image.*

This problem can be caused mainly by variations in exposure when the photos were taken. The pictures were shot with the camera on automatic exposure mode, so the same image patch can look lighter in one frame and darker in the next. When the images are stitched together, the result is a weird visible seam. Another major cause of this is differences in white balance between frames. This can be more of a problem when shooting indoor places than landscape, because in an indoor situation there are often a variety of light sources.

Normally, some stitching software tries to adjust the colour and brightness of each image to even them out and smooth out the seams, avoiding the problem. But it works well when there are only small differences between frames; if the changes are too big, it can get some unusual colours. Then, the solution is to shoot all the frames with the same exposure settings, putting the camera into manual exposure mode and choose an exposure that's in the middle of the range for the whole scene. Also, in an indoor situation, a good complementary solution is to lock the white balance setting before take the photos.

4.2.2. Lens Distortion

When original images are taken with a lot of distortion, some coordinates in the observed images are displaced away (barrel distortion) or towards (pincushion distortion) the image centre by an amount proportional to their radial distance. Distortion can be caused by a poor quality lens, by using an extra wide angle lenses, or by having very close objects in the overlap zone.

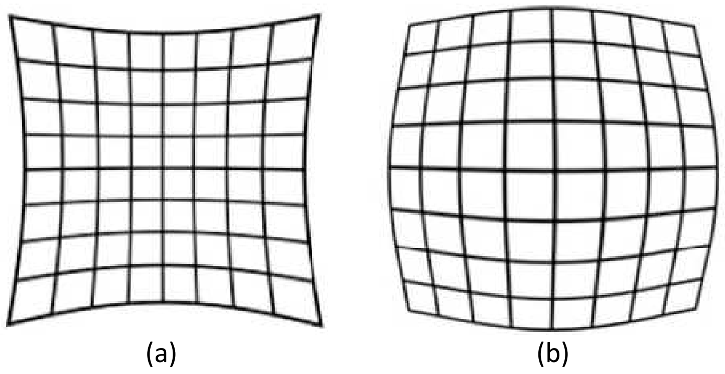


Figure 4.4. *Different types of lens distortion: (a) pincushion distortion; (b) barrel distortion.*

To reduce the problems with distortion, the local alignment is used in the image calibration stage, optimizing and adjusting distortion parameters until slightly curved lines become straight as in the original images. Another solution is by allowing plenty of overlap between input images, because lenses distort the picture more around the very edges of the frame.

4.2.3. Parallax Error

Image stitching requires that the camera rotates about the optical centre (also known as the entry pupil or no parallax point) of its lens, thereby maintaining the same point of perspective for all the photos. If the camera does not rotate about its optical centre, its images may become impossible to align perfectly; these misalignments are called parallax error. Usually, this is a problem where close objects seem to change position between frames because the camera has changed position, due to not being rotated around the entry pupil.

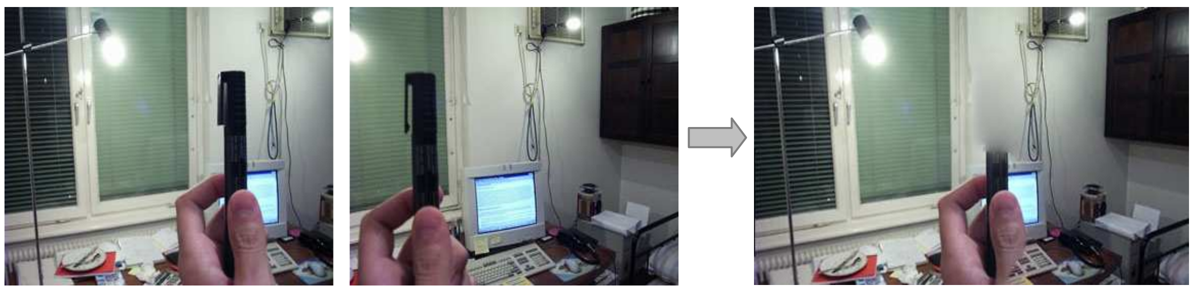


Figure 4.5. *Parallax error problem affects close objects in the final stitched image.*

The only way to avoid this error is to rotate the camera exactly around the optical point of the lens. The best way to do this is using a tripod (simple or with a head attachment), to adjust the camera position relative to the turning axis. If the photos are taken by shooting a hand-held camera, parallax error can be made undetectable holding the camera firmly, rotating the body and keeping the camera horizontal at the same height and distance.

4.2.4. Ghosting

Ghosting or rapid movement is a problem than often occurs when the scene that is being captured contains some type of movement, like people, clouds or waves. After the stitching process, it can see some ghosts or transparent double images in the areas of the photo where two adjacent pictures overlap. The main cause of this problem is when an object in the area where two frames overlap has moved position between shooting one frame and the next. However, the final result depends basically on the technique the software uses to blend the frames together.

First, before to solve ghosting problems with the software, it can try to take the photos avoiding ghosting. In a slow motion situation, to minimize the delay between shots is an efficient option. Otherwise, for fast-motion scenes, a better approach is to take more time between frames. Secondly, the software solution involves the choice of an optimal method to blend the image. Some of them have been explained in the image blending stage, such as the Laplacian Pyramid method, used in some software applications to solve this problem.

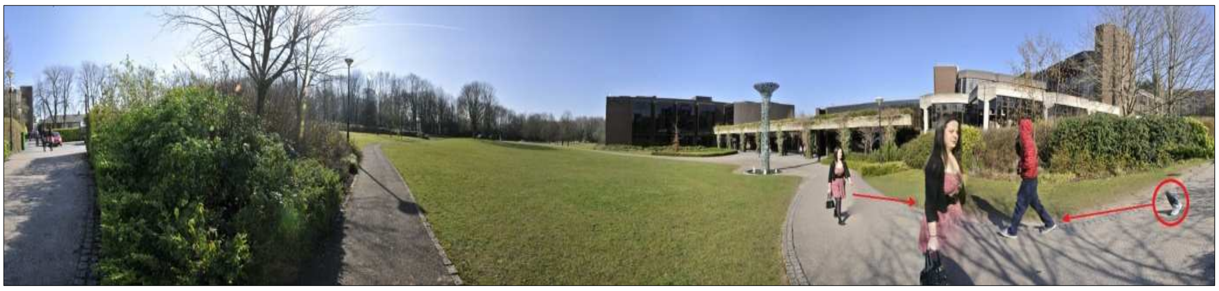


Figure 4.6. *Stitched panorama with ghosting or rapid movement problem detected.*

4.2.5. Overlapping

Many of the most common problems in image stitching are originated in the image registration stage, specifically in the keypoints detection and matching. The two main problems are vignetting and bad stitched images, a mistake in lining up the images done by the stitching software, producing a misaligned result. Both are a result of a lack of overlap between the input images.

Vignetting is observed when the picture gets noticeably darker in the very corners of the frame. Normally it is caused if the amount of overlap between frames is small, but also if the cameras used have a very wide angle lenses. One way to solve vignetting is to allow plenty of overlap between frames. Even if the individual frames of a wide angle lens are a bit dark in the corners, the dark areas will be cropped out and will not show up.

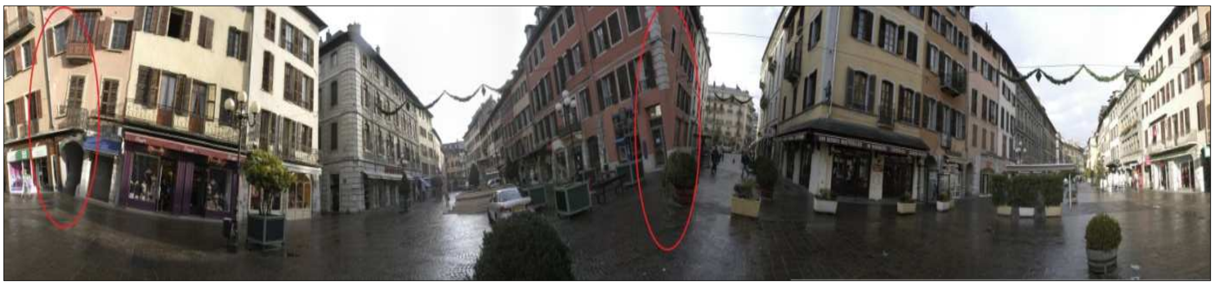


Figure 4.7. *Stitching problems in final panorama due to lack of enough overlap between frames.*

The second problem can have different causes such as lack of visible features or highly repetitive features. In the first case exists the difficulty to detect keypoints due to the picture has large areas without many obvious features (like a clear sky or blue sea). On the other case, the software can become confused if the images contain a lot of almost identical features repeated throughout the image, forming repetitive patterns across large portions of the image. Allowing an overlap of one third between frames is normally enough to solve this problem. Otherwise, some stitching software can usually solve it by manually adjusting the control points of each input image.

4.2.6. HFOV and Image Resolution

These two last issues are common when the entire stitching process is automatically executed. The Horizontal Field Of View (Hfov) information must be present to develop the stitching, and the image resolution can directly affect the delay time of the implemented application performance. The solutions for both problems require a manual interaction with the input pictures or with the application.

HFOV indicates how many degrees are contained within one image. For any stitching software program it is needed to know the HFOV of the lens of the camera used. Therefore, several pictures have Exif data (Exchangeable image file format saved with the image data by the camera) that contains different information, allowing to the stitching program automatically can detect the focal length that has been used and estimate the HFOV. The problem is when there is not Exif data, so the only way to solve it is to provide manually HFOV or a focal length and focal length multiplier for the input images in the appropriate software.

For high resolution input images, the detection and matching process is much easier due to the quantity of pixels per image. However, it means more execution time of the whole stitching process, increasing the time from a few seconds to the order of minutes. One way to do faster the execution is to take the pictures with a normal resolution or reducing the resolution of each image before the stitching, although is another way to consume time in the whole process.

4.3. Software

In terms of software, there is an extended of dedicated programs related to image stitching process. Some of the most widely used applications are *Panorama Tools*, *Hugin* and *PTGui*, and others that are taking relevance, such as *AutoStitch*, *Autopano Pro*, *Microsoft Image Composite* Editor or *CleVR Stitcher*.

4.3.1. Panorama Tools

Also known as *PanoTools*, Panorama Tools are a suite of GNU Lesser General Public License (LGPL) programs and libraries written by Professor Helmut Dersch. Panorama Tools provides an important frame work for re-projecting and blending multiple source images into immersive panoramic of many types. These libraries serve as the underlying core engine for many panorama softwares *Graphical User Interface* (GUI) front-ends.

Panorama Tools consists of a variety of components, the most relevant of them are shown below:

- *PTmender* – Panorama stitching tool which remaps, adjusts and combines arbitrary images to panoramic views.
- *PTOptimizer* – Optimizes positions and sizes of images using control point data.
- *PTblender* – Implements the rudimentary colour correction algorithm found in PTmender.
- *PTmasker* – Computes stitching masks. It implements the ability to increase depth of field by stacking images.
- *PTroller* – Takes a set of images and merges them into a single one.
- *PTcrop* – Crop an image to its outer rectangle.
- *pano13library* – The last underlying panorama library currently used by several different panorama front-ends and command line programs.

Many interactive, graphical front-ends have been developed, to make easier working and to add functionalities in Panorama Tools, such as Hugin (open source), PTGui (commercial), and along with a variety of other applications as *enblend* or *enfuse* of Andrew Mihal.

4.3.2. Hugin

Hugin is a cross-platform open source panorama photo stitching and HDR merging program developed by Pablo d'Angelo and others. It is a GUI front-end for *Panorama Tools* of Helmut Dersch. It also provides a number of additional components and command line tools, including: *autopano-sift*, *autopano-sift-C*, *panomatic* or *autopano* for automatic creation of control points; and *enblend* and *enfuse* for seamless blending of output images.

Hugin allows for the creation of control points between two images, optimization of the image transforms along with a preview window so the user can see whether the panorama is acceptable or not. There is the option of create the control points automatically. When the preview composite is correct, the panorama can be fully stitched, transformed and saved in a wide variety of standard image format.

Hugin and the associated tools have a range of advanced features:

- Combine overlapping images for panoramic photography.
- Perform advanced photometric corrections in complete panorama images, in terms of exposure, vignetting and white balance between photos.
- Stitch large mosaics of images and photos.
- Find control points and optimize parameters with the help of software assistants.
- A full range of lenses are supported, from simple camera phones to obscure fisheye lenses.
- Output several projection types, such as equirectangular, mercator, cylindrical, stereographic and sinusoidal.
- Generate HDR, exposure fused or focus stacked output from bracketed photos.
- Use 16-bit and HDR input data originally.

4.3.3. PTGui

PTGui is a commercial panoramic photo stitching software for Windows and Mac OSX. PTGui is a product developed by New House Internet Services BV, initially founded in 1996 to provide custom software development and consulting services. Originally developed as a GUI frontend to Panorama Tools, PTGui now is a full featured photo stitching and blending application.

PTGui is usually used to stitch any number of photos into a panoramic image. It can support telephoto, normal, wide angle and fisheye lenses to create partial cylindrical up to full spherical panoramas. *PTGui Pro* also includes HDR and tone mapping support.

Some of the most interesting features of PTGui program are listing below:

- PTGui can stitch multiple rows of images.
- Create 360 degrees cylindrical panoramas, flat partial panoramas and even spherical 360x180 degrees panoramas.
- No need to keep the camera level because PTGui can stitch rotated and tilted images.
- Virtually unlimited output size that involves the creation of Giga-pixel panoramas from hundreds of images.
- Layered output allows full control over the final stitched result.
- PTGui stitches most panoramas fully automatically, but at the same time provides full manual control over every single parameter. This enables stitching of difficult scenes, where other programs fail.
- Full 16 bit workflow for best image quality.

4.3.4. AutoStitch and Autopano Pro

AutoStitch is a proprietary image stitching software tool for creating panoramas, developed by Matthew Brown and David G. Lowe. This software uses SIFT and RANSAC methods, and differs from some other applications in that automatically stitches together even unaligned or zoomed photos seamlessly without user input. The only requirement is that all images be taken from a single point. A complete panorama stitching suite based on that same technology is Autopano Pro.

Autopano Pro is a commercial integrated panorama stitching software for Windows, Mac OS and Linux created by Alexandre Jenny, which automatically finds and stitches images. It is based on the AutoStitch project, and is one of the few commercial licensees of this engine.

The software has advanced features such as distortion correction, colour correction multiple blending modes, HDR support, verticals alignment, cropping and re-centring; being also useful for batch processing. An advanced version of the application is *Autopano Giga*, with more and sophisticated features.

4.3.5. Microsoft Image Composite Editor

This advanced panoramic image stitcher is developed by the research division of Microsoft Corporation. The application takes a set of overlapping photographs of a scene shot from a single camera location and creates a high resolution panorama incorporating all the source images at full resolution. The stitched panorama can be saved in a wide range of file formats, from common formats like JPEG and TIFF to multi-resolution tiled formats like HDView and Silverlight DeepZoom, as well as allowing multi-resolution upload to the Microsoft Photosynth site. Some characteristic features are that the exposure blending uses Microsoft Research fast Poisson algorithm and has no image size limitation (stitch Giga-pixel images).

4.3.6. CleVR

CleVR or Clementine Virtual Reality is a free panoramic photo sharing site and photo stitching software. The most distinguish feature of CleVR is the way it allows panoramas to be easily shared and posted on other websites using Flash viewer. This is done in a manner similar to YouTube, using a Flash based viewer that can be embedded in a web page or viewed on the CleVR site. Panoramas can be displayed with areas in the scene that can be clicked to display other content or to navigate to another scene (hotspots). This functionality is similar to Apple’s QuickTimeVR, but it allows images, text and Flash Video (FLV) to be displayed within the panorama window.

5. STITCHING APPLICATION

After studying the image stitching process and its different issues, it is time to apply this knowledge to create an application that automatically compute the entire process, using the different software available studied before. Firstly, the application is located into the Smart Tour UL project, where some previous conditions are defined. After that, the final panorama format is selected between different projections formats presented. Once these characteristics are known, the script is created and some results are shown.

5.1. Smart Tour UL Project

Smart Tour UL Project is a project of the University of Limerick, which is being developed by the Wireless Access Research group. The main project consists in an application designed for an Android Smartphone, which tracks its location using GPS and Wi-Fi systems, and calculates the optimal route to one destiny. For the moment, all this process is done only in the university campus domain, being useful for new students or visitors, showing the best routes to their destinies.

In addition to the path-finding option, the user has the possibility to obtain in specified points of the route (normally intersection of paths) a 360 degrees panorama image, helping the user to recognize the current location and the different buildings. The main idea of this option is to situate cameras around the campus, specifically in the intersection points previously selected; then to take and send different pictures to the server, where they are stitched and available to be downloaded by the phone application. The web-server used is Ubuntu 10.04 server, based on Linux distribution. This process is repeated each default time, approaching to a real time situation with different weather conditions, changes in number of people and possible works construction in the campus.

More in detail, each point location can have different number of cameras that should take enough pictures (JPEG format), allowing at least one third of overlapping between images. Afterwards, each camera sends the package of pictures to the server, where there are different folders for each intersection point. It is in this point where a stitching application is required. Having a number of inputs images in the server folder, the goal is to stitch all together creating a full 360 degrees panorama, leaving the input folder empty. Once the panorama is created, it is moved to a specific folder where the phone application can access to download the correct panorama. Normally, each model of Android Smartphone allows different resolution of the final image that must be defined in the script application. The output format of the panorama is JPEG, due to its good compression and compatibility in different phones models. In the next section, the script used for the all this stitching process is explained in detail.

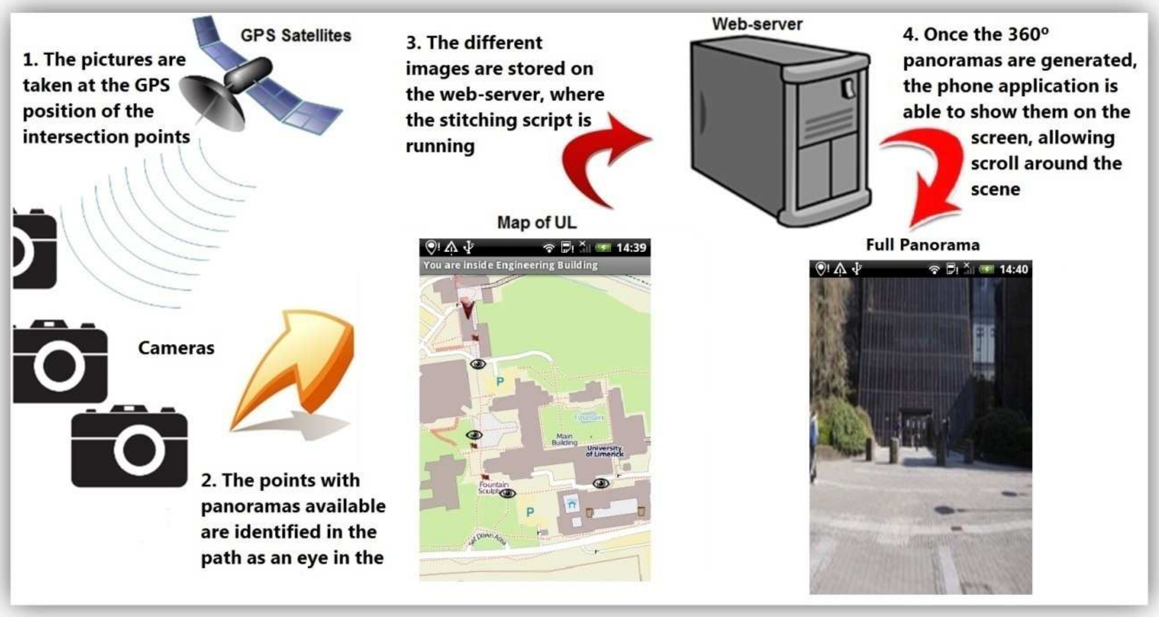

1. The pictures are taken at the GPS position of the intersection points
GPS Satellites
Cameras
2. The points with panoramas available are identified in the path as an eye in the
3. The different images are stored on the web-server, where the stitching script is running
Map of UL
Web-server
4. Once the 360º panoramas are generated, the phone application is able to show them on the screen, allowing scroll around the scene
Full Panorama
You are inside Engineering Building

Figure 5.1. *Smart Tour UL project diagram concept.*

5.2. Panorama Stitching Formats

Once it is known the goal of the final stitched image, it is time to view the different possible panoramas available, to choose the most adequate to the project. The most widely used are presented below with a few characteristics:

Rectilinear image projections have the primary advantage that they map all straight lines in three-dimensional space to straight lines on the flattened two-dimensional grid. This projection type is what most ordinary wide angle lenses aim to produce, so this is perhaps the most popular projection. Its primary disadvantage is that it can greatly exaggerate perspective as the angle of view increases, leading to objects appearing skewed at the edges of the frame. For this reason, rectilinear projections are generally not recommended for angles of view much greater than 120 degrees.

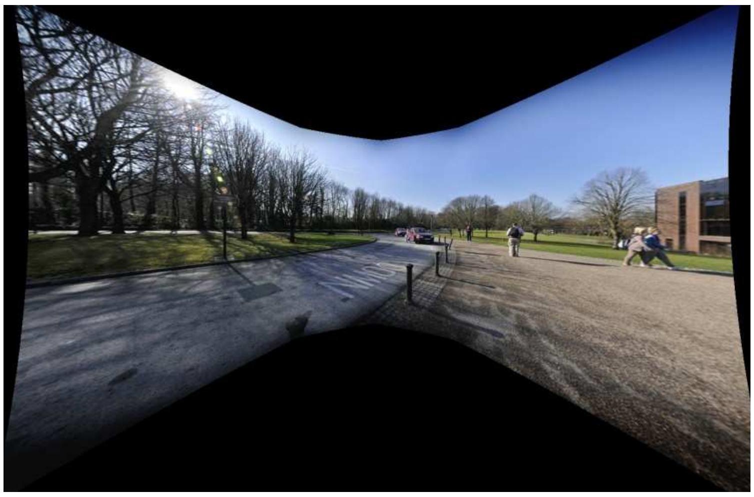


Figure 5.2. *Rectilinear projection format, showing its bad performance in the creation of stitched pictures with HFOV bigger than 120 degrees.*

Cylindrical projection is similar to equirectangular, except it also vertically stretches objects as they get closer to the north and south poles, with infinite vertical stretching occurring at the poles. It allows that the stitched image can show a 360 degrees horizontal field of view and a limited vertical field of view. It means that cylindrical projections are also not suitable for images with a very large vertical angle of view. Cylindrical projections are also the standard type rendered by traditional panoramic film cameras with a swing lens. Cylindrical projections maintain more accurate relative sizes of objects than rectilinear projections; however this is done at the expense of rendering lines parallel to the viewer's line of sight as being curved.

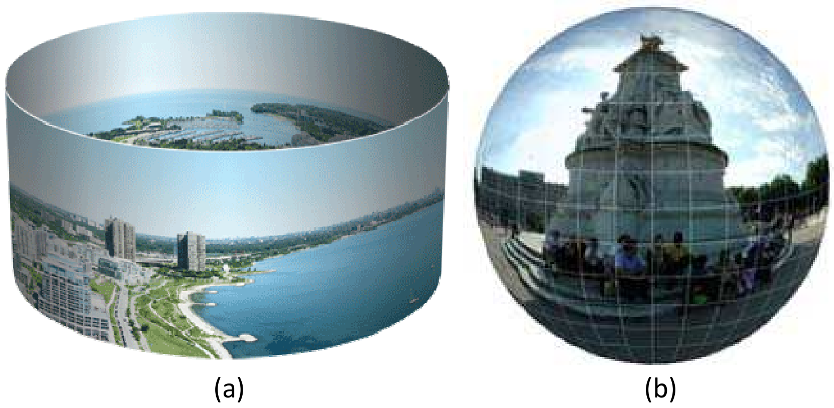


Figure 5.3. *Cylindrical and spherical projection formats: (a) Limited vertical FOV, like to be inside of a cylinder, without top and ground; (b) $\pm$ 90 degrees of VFOV, like to be inside of a sphere.*

Spherical or *Equirectangular* image projection, which is concretely one type of cylindrical projection; maps the latitude and longitude coordinates of a spherical globe directly onto horizontal and vertical coordinates of a grid, where the stitched image shows a 360 degrees horizontal by 180 degrees vertical field of view, i.e., the whole sphere. Panoramas in this projection are meant to be viewed as though the image is wrapped into a sphere and viewed from within. When viewed on a 2D plane, horizontal lines appear curved as in a cylindrical projection, while vertical lines remain vertical. These characteristics allow a good complete view of the panorama, and it is for this reason that the script application creates the output panorama in spherical projection format.

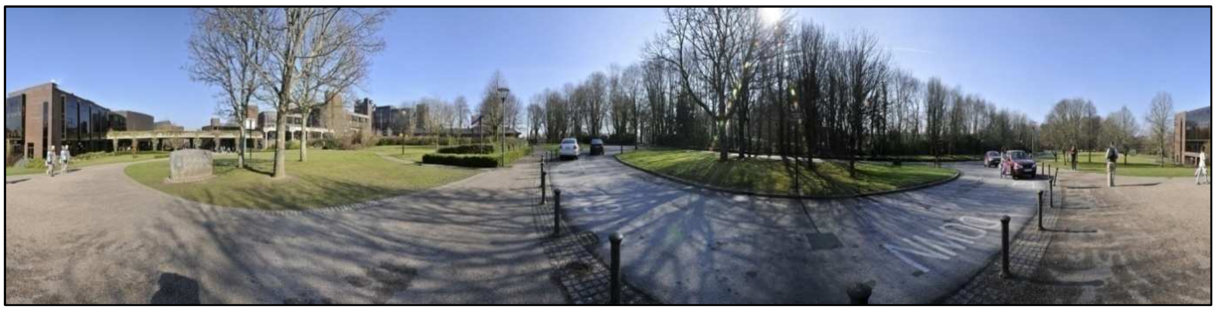


Figure 5.4. *Full 360 degrees spherical or equirectangular panorama*

Stereographic or *fisheye* projection can be used to form a little planet panorama by pointing the virtual camera straight down and setting the field of view large enough to show the whole ground and some of the areas above it. These projections are also limited to vertical and horizontal angles of view of 180 degrees or less, yielding an image which fits within a circle. These are generally not used as an output format for panoramic photography, but may instead represent the input images when the camera lens type being used for photo stitching is a fisheye lens. The difference between them is that the stereographic projection maintains a better sense of perspective by progressively stretching objects away from the point of perspective than fisheye projection.

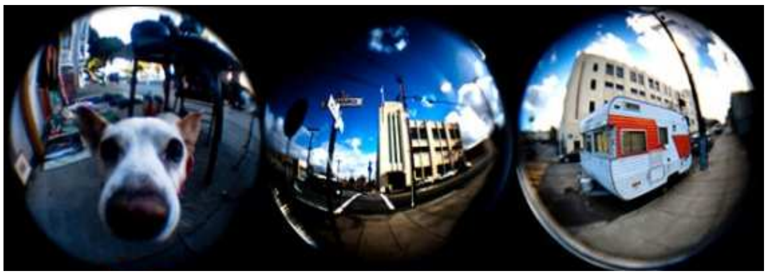


Figure 5.4. *Different picture taken with a stereographic or fish-eye lens projection format.*

Mercator image projections are most closely related to the cylindrical and equirectangular projection types; mercator represents a compromise between these two types, providing for less vertical stretching and a greater usable vertical angle of view than cylindrical, but with more line curvature. This projection is perhaps the most recognizable from its use in flat maps of the earth.

5.3. Script

In order to generate the panoramic image for Smart Tour UL project, it has been decided to create the application with open source tools, specifically working in a Linux context, with an Ubuntu server used in the main project. The idea is create a shell script with all the execution lines of the different tools used, indicating inputs, outputs, and the best options for the stitching. Moreover, it is the best option if an automatic stitching process is wanted. It means that having a number of input images of one place, executing the script, automatically generates a 360 degrees panorama. In this section, first it is explained how to get and how to use the tools and packages selected for the application. After, the final script is presented, explaining each execution line.

5.3.1. Packages and Libraries

Firstly, after to choose Ubuntu server as work environment, it has to decide the tools that are going to be used in the application. Some software has been presented in previous chapters, and comparing between them, one good approach to the project requirements is Hugin and Panorama Tools, because both of them are open sources. This allows working separately with the different commands, creating an automatic process execution application. For this reason, the script is based in Hugin performance, using some of its own tools and other complementary tools. The list of the tools with its package and libraries required is shown below:

- Hugin: The software what is based the application, and most of its tools are required. For this is needed the *Hugin 2010.4.0* version, including the command line package *hugin-tools.*
- Pano Tools: Because Hugin is essentially a GUI frontend for Panorama Tools, it is required to get both Pano Tools Utilities and Pano Tools Library. The latest versions are *libpano13-bin* and *libpano13-1*.
- Autopano-sift-C: The *autopano-sift-c* 2.5.1 version of the automatically creator of control points of the application.

- *Enblend*: Image blending tool in its 4.0 version.
- ImageMagick: The package *imagemagick* with image manipulation programs, and the libraries *libmagickcore3* and *libmagickwand3,* as interfaces between C programming language and ImageMagick image processing libraries.
- Exiftool: The *libmagickwand3* library and program to read and write metainformation in image files.

Depending on the version of Ubuntu Server, some of these tools could need other libraries or complementary packages to be installed, but it is always specified when the tools are being installing.

5.3.2. Panorama_Script.sh

The idea of the script is to create an entire panorama process performed on the command-line, and therefore to work directly with input images and project files (.pto) generated. These files are simple plain-text, sometimes it is useful and easy to modify them directly with a text editor. Obviously, this process could be done in a graphical tool, but it would not be an automatic stitching execution. In this subsection, it is presented all the tools and command lines used in the final shell script of the applications, explaining their basic functions inside the stitching process, and their performance with the options selected for the script.

For further detailed information, the final script is shown in appendix A. In addition, more information about each tool used in the final script can also be found in appendix B, where all the manual pages of the tools used are presented with different possible options.

Calling the Script

The command line to call the script can have two inputs, to clarify the location and the HFOV of the input images, this last when Exif data is missing in the single input images.

$ ./Panorama_Script.sh number_location HFOV

The inputs number of location and HFOV are saved in the variables $1 and $2 respectively.

Autopano-sift-C

It is is an automatic executable control point generator, that combines keypoint finding and matching of a group of overlapping input images, and is based on the SIFT algorithm. The input image files can have any format (JPEG, PNG, TIFF...), and the output is a panorama project file.

The number of inputs can be from two to more images, and it is highly recommended to be all of the same dimension and scale. Once defined the inputs, the SIFT detection is performed, following by the RANSAC filtering and the fast kd-tree NN search implementation for keypoint matching. Autopano-sift-c uses a smooth method for reducing image size, which leads to stable control point positions. Some notable options are the control of the limit on the larger image dimension (1600 by default), and the maximum number of control points per image pair (25 by default). Also the projection format and the HFOV of the input images should be written, usually selecting 0 for a rectilinear input projection.

The command line for a complete control point detection and matching, allowing RANSAC, kd-tree NN search and unlimited control points per image pair is the following:

```
$ autopano-sift-c --ransac on --maxmatches 0 --projection 0,$2 project.pto input_images[...]
```

Input*: input_images (JPEG)*
Output*: project.pto*

Celeste_Standalone

Once the project file is created, it is time to clean up some not useful control points of the file. The reason of using this tool is because of control point generators are not very good at distinguishing between static objects and objects that change quickly such as clouds. This causes alignment problems, because clouds can easily move several pixels between shots. Celeste_standalone is a hugin-tool, which works with sky areas, recognizing clouds. Using support vector machine (SVM) techniques, it identifies clouds in the input photos and removes control points from these areas. The control points over one default threshold value are deleted from the input project file (-i), creating other as output adding _celeste.pto by default:

```
$ celeste_standalone -i project.pto
```

Input*: project.pto*
Output*: project_celeste.pto*

CPclean

In addition to the control points in sky areas, sometimes there are some wrong control points, with large error distances. Cpclean removes all these bad control points by statistical filter methods, working with mean and standard deviation for all images pairs and also all the whole panorama. It deletes by default all the control points that have an error bigger than $mean + 2standard\_deviation$, overwriting the project file adding _clean.pto:

```
$ cpclean project_celeste.pto
```

Input*: project_celeste.pto*
Output*: project_celeste_clean.pto*

Autooptimiser

Up to now, the project file only contains a simple list of good control points and input images, so the images are not yet aligned. This step is done by geometric optimization of images positions, updating the .pto file. Autooptimiser aligns and optimizes position and geometry in the project file through different options, such as auto align mode (-a), level horizon (-l) and automatic selection of suitable output projection and size (-s). An example of input project file appears in appendix C.1, with all the diferents options.

```
$ autooptimiser -a -l -s -o optimized.pto project_celeste_clean.pto
```

Input*: project_celeste_clean.pto*
Output*: optimized.pto*

Pano_modify

Before start with the main stitching process, the pano_modify hugin-tool allows to change some output options of the project file on the command line. It can set the output panorama projection, commonly cylindrical (1) or spherical (2) formats, but in the case of this application the spherical is selected. Often, it permits also to calculate the optimal field of view when the camera lens information is provided by the input images. Other interesting options are to straighten (-s) and to centre (-c) the panorama, calculate the optimal canvas size, and to set automatically the optimal crop rectangle of the final result.

```
$ pano_modify --projection=2 --fov=AUTO -s -c --canvas=AUTO --crop=AUTO optimized.pto
```

Input*: optimized.pto*
Output*: optimized_mod.pto*

Pto2mk and Make

Once the project file has been created, optimized and modified, it is prepared to start the stitching. The way to start the stitching is generating a makefile that contains all the rules to stitch and blend the images. Pto2mk generates the output makefile (-o) from the existing panorama project file, with a prefix (-p) name. After, the commands make processes this makefile, with the tools nona, enblend and exiftool, deleting all the intermediate files created once finished:

```
$ pto2mk -o final.pto.mk -p final optimized_mod.pto
$ make -e -f final.pto.mk all clean
```

Input*: optimized_mod.pto*
Output*: final.pto.mk*

Nona

It is a stitch hugin-tool, where no seam feathering is done. Nona uses functions of Pano Tools to remap all the images into a panorama. Nona performs geometrical and photometric distortions on photos and writes the output to image files. However nona doesn't decide what the distortions are going to be, it just does the *remapping* part of the stitching process. The parameters to apply in nona are specified in the final project file, which has been modified by all the previous stages. Appendix C.2 shows an input project file for nona.

Multiple TIFF files (TIFF_m) are chosen to be the output format (-m), due to their easy performance in the next blending stage. The compression type (-z) selected for TIFF output is LZW, and *ldr* (low dynamic range) output mode (-r) to keep the original bit depth and response of the images. Nona command-lines are automatically generated in the makefile, and any modification is required. The output images are now prepared to be blended:

```
$ nona -z LZW -r ldr -m TIFF_m -o final optimized_mod.pto input_images[...]
```

Input*: input_images (JPEG), optimized_mod.pto*
Output*: multiple_final_images (TIFF)*

Enblend

Enblend overlays and combines multiple TIFF images trying to make the seams between the input images invisible. This tool blends the images in the order specified on the command line, according to the way that they are overlap. Following with the same kind of compression, LZW is used for the output image. For a full 360 degrees output panorama is useful to blend around the -180/+180 degrees boundary (-w). The final size of the output image has been written by pano_modify previously. As nona, enblend command line is generated in the makefile just after nona lines:

```
$ enblend --compression=LZW -w -o final.tif multiple_final_images.tif[...]
```

Input*: multiple_final_images (TIFF)*
Output*: final (TIFF)*

Convert

Once the final panorama is completely stitched, it is time to prepare the image for the final device. In the project, the final device is an android phone that will visualize the full panorama on the screen. The best option is to convert the TIFF file to JPEG, due to its features of size and compatibility. Depending on the phone model used, the image size in pixels should be modified too. Sometimes, the stitched image results rotated 180 degrees, and also a rotation is required. ImageMagick package presents the command line convert, which allows these conversions in the final panorama:

```
$ convert –resize widthxheight\! –rotate 180 final.tif pano$1.jpg
```

Input*: final (TIFF)*
Output*: pano$1 (JPEG)*

Exiftool

Finally, it is interesting to keep the Exif data in the final result, as GPS location, date and time, camera model, author and other remarkable information. First, this information is copied from one input image to the final.tif image, specified in the makefile after the enblend line. But, if some conversion is applied after, as convert line, then the Exif data must be copied to the last result. Exiftool is used for this task:

```
$ exiftool -overwrite_original_in_place -TagsFromFile final.tif -ImageDescription -Make -Model -Artist
-WhitePoint -Copyright -GPS:all -DateTimeOriginal -CreateDate –UserComment -ColorSpace -
OwnerName -SerialNumber pano$1.jpg
```

Remove and Move

After all the command lines, the final script is completed, only a few additions are required to adjust the application to the project context. The proposal is to receive the input

JPEG images in a server folder, execute the script, and obtain a final panorama in JPEG format. For this reason, the folder only has to contain the inputs at the beginning, and the panorama at the end. All the inputs and intermediate files must be deleted at the end of the process. The final panorama is moved to the place where the phone can download the image when is using the Smart Tour UL application, with the name corresponding to the UL location number where the pictures were taken, previously agreed. The lines that solve these problems are written with remove and move commands:

```
$ mv pano$1.jpg /home/smart-tour /www/
$ rm *.tif *.JPG *.pto *.mk
```

5.4. Results

At this point, it can verify the performance and behaviour of the stitching script. In this section different visual examples are given as results, beginning with a simple re-stitching of an original photo. Stitched images created from normal phone camera pictures are also presented in terms to show other common case. Finally, full panoramas of the University of Limerick are created giving an approximation of what the end user can visualize.

5.4.1. Original Image Re-Stitched

One simple way to test the script is cutting one original photo from a digital camera in two parts, allowing at least one third of overlapping between both pictures. Then, they can be stitched together applying the stitching script, resulting in a seamless re-stitched photo, only losing a part of the original image in the right side of the photo, with perfect alignement.

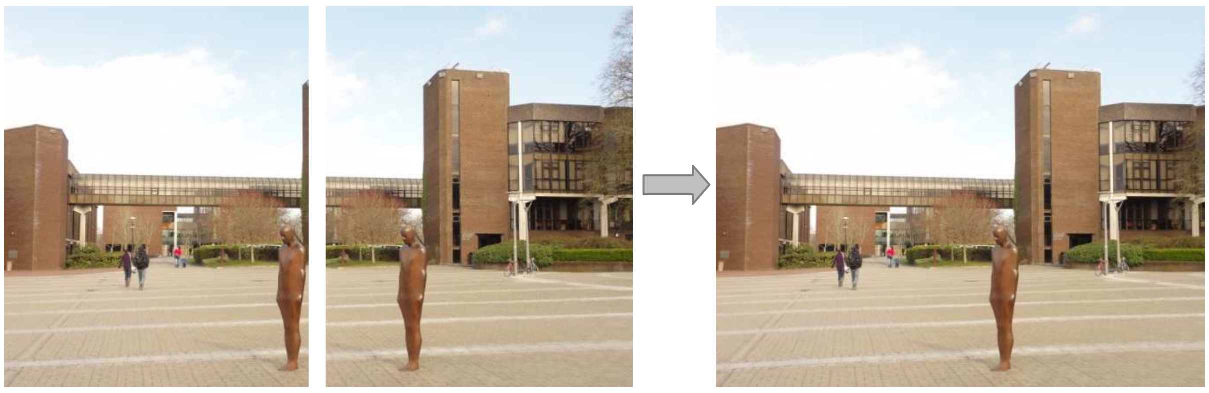


Figure 5.5. *Re-stitching an image of digital camera: (a) original image cut into two overlapped parts; (b) re-stitched image aligned with no visible seams.*

5.4.2. Camera Phone

If the photos are taken from a camera of a Smartphone, the stitching result depends also in the rotation camera level done while the pictures were taken. Precisely, for full 360 degrees panoramas, it is difficult to keep holding at the same level of rotation, often producing parallax error and lost of VFOV information. For this reason the results of the script are better for non-full panorama images. One example with two photos and other with six inputs images from an 8.1 Mega-pixel Sony Ericcson Aino camera phone are shown below:

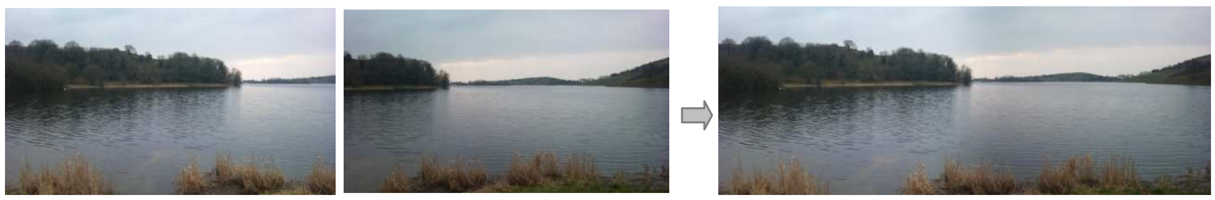


Figure 5.6. *Two overlapping photos from a Sony Ericcson Aino camera phone merged into a perfect seamless stitched image.*

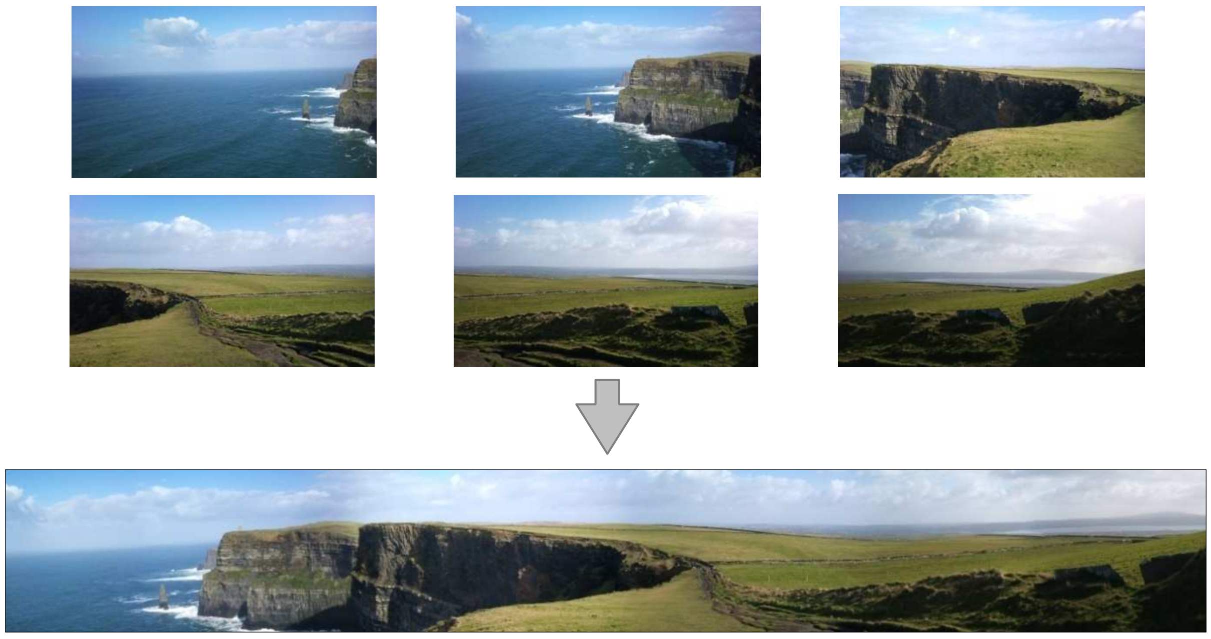


Figure 5.7. *Six photos taken with a camera phone with the same HFOV and size. The equirectangular result panorama with less than 180 degrees of HFOV has lost image information in both right and left sides. It also appears changes in exposure between the second and third frame.*

5.4.3. University Of Limerick

For the Smart Tour UL project, some photos have been taken at different locations of the main campus, using a tripod and allowing overlapping between frames with a known HFOV. After stitching, the result panoramas can be prepared for be visualized in the user Smartphone, modifying the format and size image, depending on the phone model. Lastly, the phone application can show the spherical panorama through scrolling its touch screen, navigating throughout the panoramic image of the specific location of the campus, giving the feeling of being in the exact place and helping for orientation and routing problems.

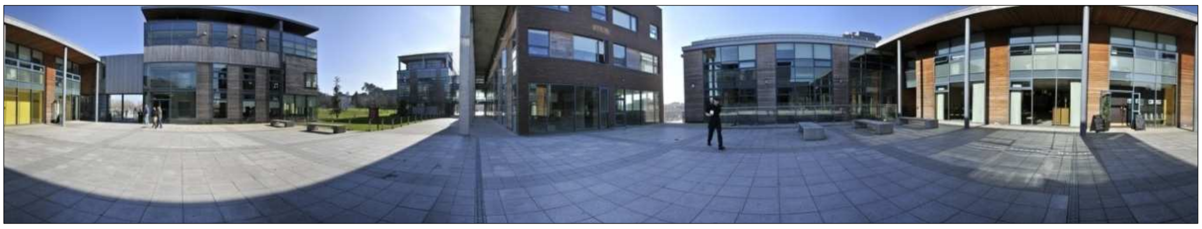


Figure 5.8. *Final full spherical panorama just after the stitching process, with its default size image.*

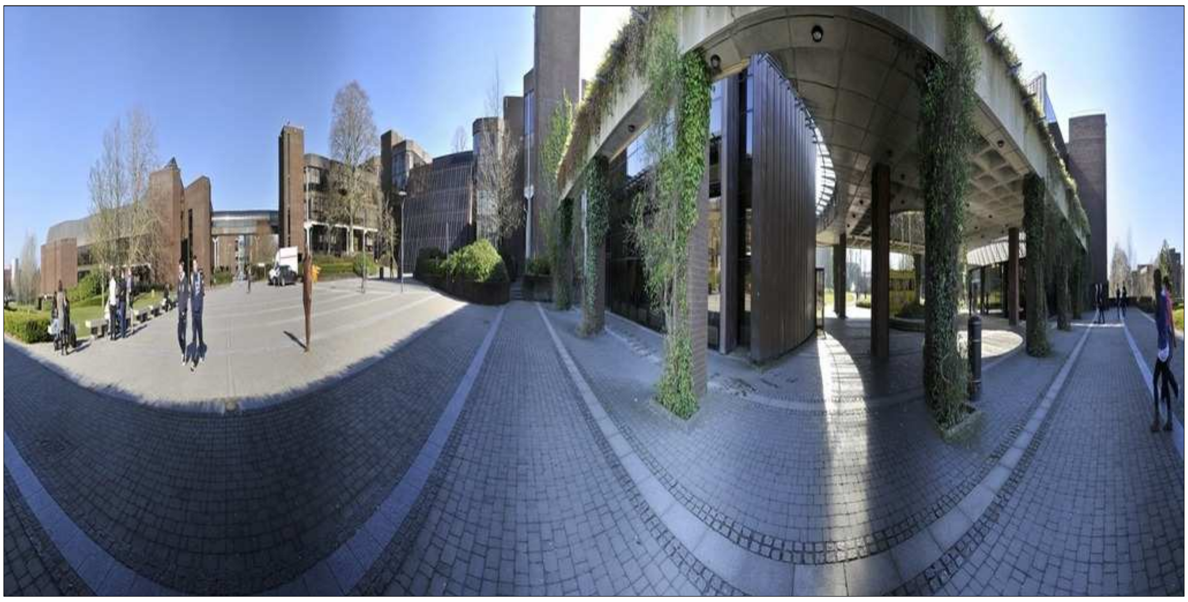


Figure 5.9. *Full spherical panorama resized to HTC Smartphone suitable resolution (1024x512pixels).*

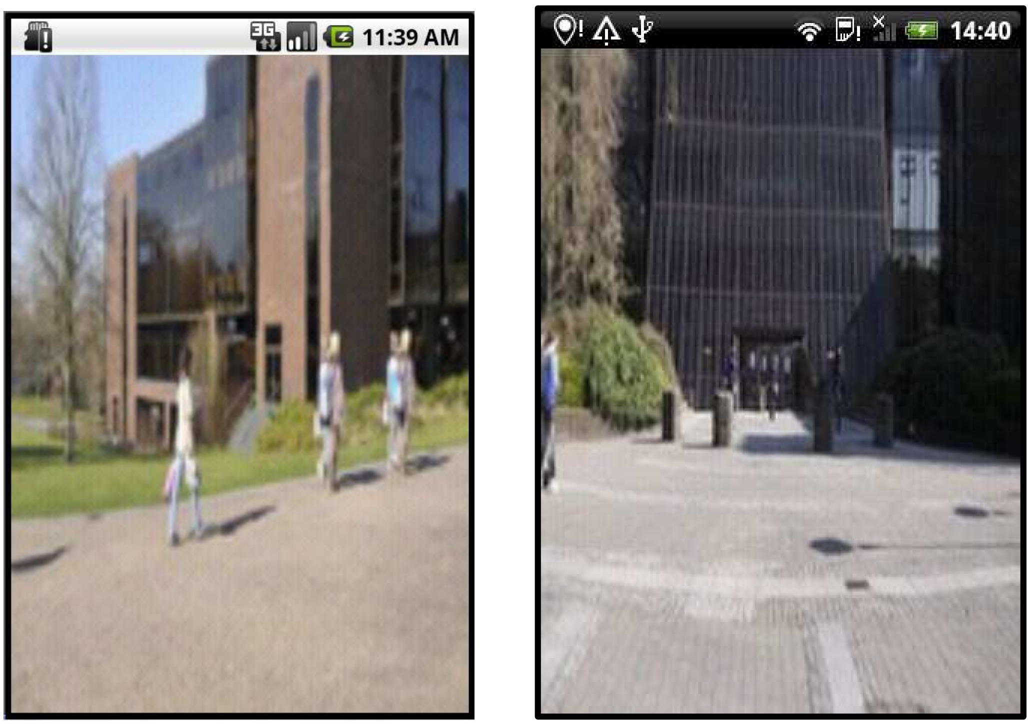


Figure 5.10. *Two different Smartphone views of two spherical panoramas, showing two different locations of the UL campus with buildings and students. Scrolling the touch screen allows to explore the entire panorama scene.*

6. CONCLUSIONS

The image stitching script presented in this document has been developed for the Smart Tour UL project, with the purpose to stitch automatically in one web server input images taken from specific locations in the UL campus. Once the full 360 degrees spherical stitched images are created, the Android Smartphone application is able to download these panoramas from the server. Finally, the final user can visualize the panorama of the specified location, allowing full horizontal field of view and 180 degrees of vertical view.

While the optimization and blending stages of the stitching process have been performed by using the open source software belonging to Hugin and Panorama Tools, the feature detection and matching have been done using the SIFT algorithm. SIFT has been chosen due to the large number of features from the images used, reducing contribution of errors caused by variations in the average error of all feature matching errors. In addition, this algorithm is invariant to scale, orientation, and affine distortion, and partially invariant to illumination changes. The registration stage with SIFT is complemented with RANSAC and Kd-Tree NN Search techniques, obtaining a suitable behaviour and good results working with overlapping images.

After several tests realized with the script in different conditions, it can conclude that the image stitching script has a correctly automatic performance, provided there is enough overlap between images, the HFOV is specified on the images or on the script, and parallax error is prevented taking pictures with a good rotation level. However, sometimes these problems can result into stitched images with only few errors or misalignments that do not affect too much in the final image.

6.1. Future Work Development

The image stitching process in this project attempts to be always automatic, optimal and precise. One possible way to ensure these conditions will be creating a project file template of each camera location. It will require that the cameras must take identical photos each time, allowing the same level of rotation, HFOV and number of pictures. However, some previous tests should be performed in order to solve possible stitching problems during the registration and optimization stages, and leaving prepared the project file for the last stage of blending. It is a good approach to optimize time of execution and to ensure a better performance of the script, first modifying the file and then letting it run automatically.

In order to improve the algorithms used in the script. A new goal could be to classify different input images according to the feature detection and matching points. Normally, the input images are connected between control points in the registration stage, discarding unconnected images. However, if there are amount of pictures of different places, it should be the option to separate accurately these images according to the feature matches between them, and then prepare them to the rest of the stitching process. In terms of solving rapid movement or scene motion problems in the final stitched images, the blending stage could be improve trying to select better how to render the seams of the images. This selection will depend on if motion image points are presented or not in the seams, to avoid cut objects in the final image.

In the case of wanting to perform the stitching process in the Smartphone device, the image compression formats and image scales have to be considered. Due to a limited memory of the device, the image scale during the process must be pre-defined and the formats to work inside the different algorithms should be compatible with the phone formats. For those reasons, the algorithms should be prepared with format conversions and scales modifications of the images, to keep execution time, quality and memory.

Finally, other possible fields to research are related to video stitching, because it is a kind of simple generalization of multiple image stitching. For example, the creation of stitched panoramas from video sources of moving cameras, so that enough overlapping is obtained of the different video frames. Moreover, objects in the foreground from the same video scene could be extracted and composited into stitched image, giving the impression of a sequential scene image. The main problems to deal with in video stitching are often the presence of large independent movement scenes and the different fields of view caused by changing the cameras focus.

7. REFERENCES

[1] d'Angelo, Pablo *Hugin homepage,* <http://hugin.sourceforge.net/>.

[2] Bolles, Robert C.; Fischler, Martin A. "*Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*" from magazine Communications of the ACM Volume 24 Issue 6, June 1981.

[3] Brown, Matthew; Lowe, David G. "*Automatic Panoramic Image Stitching using Invariant Features*", University of British Columbia, Vancouver, Canada, 2007.

[4] Brown, Matthew; Lowe, David G., *AutoStitch website,* <http://cvlab.epfl.ch/~brown/autostitch/autostitch.html#autostitch>.

[5] Chandran, Sharat "*Introduction to kd-trees*", University of Maryland Department of Computer Science. <http://www.cs.umd.edu/class/spring2002/cmsc420-0401/pbasic.pdf>.

[6] CleVR Ltd., *CleVR website,* <http://www.clevr.com >.

[7] Community of the PanoTools Yahoo Group *Panotools Next Generation*, <http://wiki.panotools.org.>.

[8] Knight, Denis "*The Perfect Panorama*", October 2008.

[9] Lowe, David G. "*Distinctive Image Features from Scale-Invariant Keypoints*", University of British Columbia, Vancouver, Canada, January 2004.

[10] McHugh, Sean "*Cambridge in Colour"* website, Photo Stitching & Digital Panoramas section <http://www.cambridgeincolour.com/>

[11] Microsoft Research, *Microsoft Research Image Composite Editor*, <http://research.microsoft.com/en-us/um/redmond/groups/ivm/ICE/>.

[12] New House Internet Services BV, *PTGUI homepage*, <http://www.ptgui.com/>.

[13] Szielski, Richard "*Computer Vision: Algorithms and Applications*", September 2010. <http://szeliski.org/Book>.

[14] Szielski, Richard "*Image Alignment and Stitching: A Tutorial*", Technical Report, December 2006.

APPENDIX A. Panorama_Script.sh

```
# Panorama Stitching Script File

autopano-sift-c --ransac on --maxmatches 0 --projection 0,$2
project.pto *.JPG

celeste_standalone -i project.pto

cpclean project_celeste.pto

autooptimiser -a -l -s -o optimised.pto project_celeste_clean.pto

pano_modify --projection=0 --fov=AUTO -s -c --canvas=AUTO
--output=modified.pto optimised.pto

pto2mk -o panorama.pto.mk -p panorama modified.pto

make -e -f panorama.pto.mk all clean

convert -resize 1024x512\! panorama.tif pano$1.jpg

exiftool -overwrite_original_in_place -TagsFromFile panorama.tif -
ImageDescription -Make -Model -Artist -WhitePoint -Copyright -
GPS:all -DateTimeOriginal -CreateDate -UserComment -ColorSpace -
OwnerName -SerialNumber pano$1.jpg

mv $1.jpg /home/oscar/Escritorio/Formats/

rm  *.pto *.mk *.tif *.JPG
```

APPENDIX B. Manual Pages

```
NAME
       autooptimiser - Optimize image positions

SYNOPSIS
       autooptimiser [options] project.pto

       (- can be specified to read the project from stdio.)

DESCRIPTION
       autooptimiser works similarly to PToptimizer but with extra
command-line options.

       It is also different in that PToptimizer appends its output
onto the input file in the form of 'o' lines which need further
processing.  Autooptimiser simply updates the project and writes it
to a new file.

OPTIONS
       -o file.pto
           Output file. If omitted, stdout is used.

       Optimisation options (if not specified, no optimisation takes
place):

       -a  Auto align mode, includes various optimisation stages,
depending on the amount and distribution of the control points

       -p  Pairwise optimisation of yaw, pitch and roll, starting
from first image

       -n  Optimize parameters specified in script file (like
PToptimizer).

       Post-processing options:

       -l  Level horizon (works best for horizontal panoramas)

       -s  Automatically select a suitable output projection and
size

       Other options:

       -q  Quiet operation (no progress is reported)

       -v HFOV
           Specify horizontal field of view of input images. Used if
the .pto file contains invalid HFOV values (autopano-SIFT writes
.pto files with invalid HFOV)

       When -a, -l, and -s options are used together, an operation
similar to the one of "Align" button in hugin is performed.
```

```
AUTHORS
       Written by Pablo d'Angelo. Also contains contributions from
Douglas Wilkins, Ippei Ukai, Ed Halley, Bruno Postle, Gerry
Patterson and Brent Townshend.

       This man page was written by Cyril Brulebois
<cyril.brulebois@enst-bretagne.fr> and is licensed under the same
terms as the hugin package itself.
```

```
NAME
     autopano-sift-c: Find control points giving a Hugin project
file

SYNOPSIS
     autopano-sift-c [options] output.pto image1 image2 [..]
          hugin project file output.pto gets the results
          input images can be jpeg, tiff, or other formats.

     autopano-sift-c [options] output.pto projectfile
          projectfile: a hugin project or other PT compatible
          script with image file names, projections and angular
          widths.  Enables stereographic projection.

DESCRIPTION
     Options
       --ransac <on|off|1|0>   Switch RANSAC filtration on or off
(default: off).
       --maxmatches <matches>  Output no more that this many control
points per
                               image pair (default: 25, zero means
unlimited)
       --maxdim <n>            Make largest image dimension <= n
(default: 1600).
       --projection <n>,<d>    n = PT format code, d = hfov in
degrees.  These
                               apply to all images, reprojection is
enabled.
       --ANNmatch <on|off|1|0> Use fast keypoint matching tree
(default: on).
       --disable-areafilter    Do not use max-area filtration, which
is default.
                               See manpage for details.
       --integer-coordinates   Truncate match coordinates to integer
numbers.
       --absolute-pathnames <on|off|1|0>   Use the absolute pathname
of the image
                               file in the PTO output file. Disabled
by default.

     Alignment options
       --align                 Automatically pre-align images in PTO
file.
       --bottom-is-left
       --bottom-is-right       Use in case the automatic algorithm
fails.
       --generate-horizon <c>  Generate up to 'c' horizon lines.

     Refinement options
       --refine                Refine the found control points using
the
                               original images.
       --refine-by-middle      Use the best middle point to refine
(default).
       --refine-by-mean        Use the mean of the patches control
points.
```

```
       --keep-unrefinable <on|off|1|0>
                               Keep unrefinable matches (default:
on).
     output.pto: The output PTO panorama project file.
          The filename can be "-", then stdout is used
          image<n>: input image files (any common format: JPEG,
          PNG, TIFF, ..)

     Notice: for the aligning to work, the input images shall be
       1. All of the same dimension and scale
       2. The first images must be an ordered row.

==================================================================
The use of this software is restricted by certain conditions.
See the "LICENSE" file distributed with the program for details.

The University of British Columbia has applied for a patent on the
SIFT algorithm in the United States.  Commercial applications of
this software may require a license from the University of British
Columbia.
=================================================================
NAME
       celeste_standalone - Cloud identification

SYNOPSIS
       celeste_standalone [options] image1 image2 [..]

DESCRIPTION
       Celeste has been trained using Support vector machine
techniques to identify clouds in photos and remove control points
from these areas.
       celeste_standalone is a command-line tool with all the same
functionality as Celeste in hugin.

       Simple usage is to just 'clean' an existing project file:

        celeste_standalone -i project.pto -o project.pto

OPTIONS
       -i <filename>
           Input Hugin PTO file. Control points over SVM threshold
will be removed before being written to the output file. If -m is
set to 1, images in the file will be also be masked.

       -o <filename>
           Output Hugin PTO file. Default:'<filename>_celeste.pto'

       -d <filename>
           SVM model file. Default: 'data/celeste.model'

       -s <int>
           Maximum dimension for re-sized image prior to processing.
A higher value will increase the resolution of the mask but is
significantly slower. Default:
           800

       -t <float>
           SVM threshold. Raise this value to remove fewer control
points, lower it to remove more. Range 0 to 1. Default: 0.5

       -m <1|0>
           Create masks when processing Hugin PTO file. Default: 0

       -f <string>

       -r <1|0>
           Filter radius. 0 = large (more accurate), 1 = small
(higher resolution mask, slower, less accurate). Default: 0

       -h  Print usage.

AUTHORS
       Written by Tim Nugent.
```

```
NAME
       convert - convert between image formats as well as resize an
image, blur, crop, despeckle, dither, draw on, flip, join, re-
sample, and much more.

SYNOPSIS
       convert [input-options] input-file [output-options] output-
file

OVERVIEW
       The  convert  program  is  a  member  of the ImageMagick(1)
suite of tools.  Use it to convert between image formats as well as
resize an image, blur, crop, despeckle, dither, draw on, flip, join,
re-sample, and much more.

DESCRIPTION
       Image Settings:
         -depth value          image depth
         -size geometry        width and height of image
         -resize geometry      resize the image
         -rotate degrees       apply Paeth rotation to the image

       By  default,  the image format of `file' is determined by its
magic number.  To specify a particular image format, precede the
filename with an image format name and a colon (i.e. ps:image) or
specify the image type as the filename suffix (i.e. image.ps).
Specify 'file' as '-' for standard input or output.

COPYRIGHT
       Copyright © 1999-2010  ImageMagick  Studio  LLC.  Additional
copyrights  and  licenses  apply  to  this  software.
```

```
NAME
       cpclean - Remove wrong control points by statistical methods

SYNOPSIS
       cpclean [options] input.pto

DESCRIPTION
       cpclean uses statistical methods to remove wrong control
points.

       Step 1 optimises all images pairs, calculates for each pair
mean and standard deviation and removes all control points with
error bigger than mean+n*sigma.

       Step 2 optimises the whole panorama, calculates mean and
standard deviation for all control points and removes all control
points with error bigger than mean+n*sigma.

OPTIONS
       -o output.pto
           Output Hugin PTO file. Default: '<filename>_clean.pto'.

       -n num
           distance factor for checking (default: 2)

       -p  do only pairwise optimisation (skip step 2)

       -w  do optimise whole panorama (skip step 1)

       -h  shows help

AUTHORS
       Thomas Modes
```

```
NAME
       enblend - combine images using a multiresolution spline

SYNOPSIS
       enblend [options] [--output=IMAGE] INPUT...

DESCRIPTION
       Blend INPUT images into a single IMAGE.

       INPUT... are image filenames or response filenames.  Response
filenames start with an "@" character.

   Common options:
       -V, --version
              output version information and exit

       -a     pre-assemble non-overlapping images

       -h, --help
              print this help message and exit

       -l, --levels=LEVELS
              number of blending LEVELS to use (1 to 29); negative
number of LEVELS decreases maximum

       -o, --output=FILE
              write output to FILE; default: "a.tif"

       -v, --verbose[=LEVEL]
              verbosely report progress; repeat to increase
verbosity or directly set to LEVEL

       -w, --wrap[=MODE]
              wrap around image boundary, where MODE is NONE,
HORIZONTAL, VERTICAL, or BOTH; default: none; without argument the
option selects horizontal wrapping

       -x     checkpoint partial results

       --compression=COMPRESSION
              set compression of output image to COMPRESSION, where
COMPRESSION is: NONE, PACKBITS, LZW, DEFLATE for TIFF files and 0 to
100 for JPEG files

   Extended options:
       -b BLOCKSIZE
              image cache BLOCKSIZE in kilobytes; default: 2048KB

       -c     use CIECAM02 to blend colors

       -d, --depth=DEPTH
              set the number of bits per channel of the output
image, where DEPTH is 8, 16, 32, r32, or r64

       -g     associated-alpha hack for Gimp (before version 2) and
Cinepaint
```

```
       --gpu  use graphics card to accelerate seam-line optimization

       -f WIDTHxHEIGHT[+xXOFFSET+yYOFFSET]
              manually set the size and position of the output
image; useful for cropped and shifted input TIFF images, such as
those produced by Nona

       -m CACHESIZE
              set image CACHESIZE in megabytes; default: 1024MB

   Mask generation options:
       --coarse-mask[=FACTOR] shrink overlap regions by FACTOR to
speedup mask generation; this is the default; if omitted FACTOR
defaults to 8

       --fine-mask
              generate mask at full image resolution; use e.g.  if
overlap regions are very narrow

       --smooth-difference=RADIUS
              smooth the difference image prior to seam-line
optimization with a Gaussian blur of RADIUS; default: 0 pixels

       --optimize
              turn on mask optimization; this is the default

       --no-optimize
              turn off mask optimization

       --optimizer-weights=DISTANCEWEIGHT[:MISMATCHWEIGHT]
              set the optimizer's weigths for distance and mismatch;
default: 8:1

       --mask-vectorize=LENGTH
              set LENGTH of single seam segment; append "%" for
relative value; defaults: 4 for coarse masks and 20 for fine masks

       --anneal=TAU[:DELTAEMAX[:DELTAEMIN[:KMAX]]]
              set annealing parameters of optimizer strategy 1;
defaults: 0.75:7000:5:32

       --dijkstra=RADIUS
              set search RADIUS of optimizer strategy 2; default: 25
pixels

       --save-masks[=TEMPLATE]
              save  generated  masks  in TEMPLATE; default: "mask-
%n.tif"; conversion chars: %i: mask index, %n: mask number, %p: full
path, %d: dirname, %b: baseâ€
              name, %f: filename, %e: extension; lowercase
characters refer to input images uppercase to the output image

       --load-masks[=TEMPLATE]
```

```
              use existing masks in TEMPLATE instead of generating
them; same template characters as "--save-masks"; default: "mask-
%n.tif"

       --visualize[=TEMPLATE] save results of optimizer in TEMPLATE;
same template characters as "--save-masks"; default: "vis-%n.tif"

AUTHOR
       Written by Andrew Mihal and others.

REPORTING BUGS
       Report bugs at <http://sourceforge.net/projects/enblend/>.

COPYRIGHT
       Copyright © 2004-2009 Andrew Mihal.  License GPLv2+: GNU GPL
version 2 or later <http://www.gnu.org/licenses/gpl.html>
       This is free software: you are free to change and
redistribute it.  There is NO WARRANTY, to the extent permitted by
law.
```

```
NAME
       exiftool - Read and write meta information in files

SYNOPSIS
       exiftool [OPTIONS] [-TAG...] [--TAG...] FILE...

       exiftool [OPTIONS] -TAG[+-<]=[VALUE]... FILE...

       exiftool [OPTIONS] -tagsFromFile SRCFILE [-
SRCTAG[>DSTTAG]...] FILE...

       exiftool [ -ver | -list[w|f|wf|g[NUM]|d|x] ]

DESCRIPTION
       A command-line interface to Image::ExifTool, used for reading
and writing meta information in image, audio and video files.  FILE
is a source file name, directory name, or "-" for the standard
input.  Information is read from the source file and output in
readable form to the console (or written to an output text file with
the -w option).

OPTIONS
       Case is not significant for any command-line option
(including tag and group names), except for single-character options
when the corresponding upper-case option is defined.  Many single-
character options have equivalent long-name versions (shown in
brackets), and some options have inverses which are invoked with a
leading double-dash.  Note that multiple single-character options
may NOT be combined into one argument because this would be
interpreted as a tag name.

   Option Summary
         -c FMT      (-coordFormat)       Set format for GPS
coordinates
         -d FMT      (-dateFormat)        Set format for date/time
values
         -overwrite_original_in_place     Overwrite original by
copying tmp file
         -tagsFromFile SRCFILE            Copy tag values from file
(ImageDescription, Model, Artist, WhitePoint, Copyright,GPS
DateTimeOriginal, CreateDate, UserComment, Colorspace, OwnerName,
SerialNumber

DIAGNOSTICS
       The exiftool application exits with a status of 0 on success,
or 1 if an error occured or if all files failed the -if condition.

AUTHOR
       Copyright © 2003-2010, Phil Harvey. This is free software;
you can redistribute it and/or modify it under the same terms as
Perl itself.
```

```
NAME
       mv - move (rename) files

SYNOPSIS
       mv [OPTION]... [-T] SOURCE DEST
       mv [OPTION]... SOURCE... DIRECTORY
       mv [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

       Mandatory arguments to long options are mandatory for short
options too.

       --backup[=CONTROL]
              make a backup of each existing destination file

       -b     like --backup but does not accept an argument

       -f, --force
              do not prompt before overwriting

       -i, --interactive
              prompt before overwrite

       -n, --no-clobber
              do not overwrite an existing file

       If you specify more than one of -i, -f, -n, only the final
one takes effect.

       --strip-trailing-slashes
              remove any trailing slashes from each SOURCE argument

       -S, --suffix=SUFFIX
              override the usual backup suffix

       -t, --target-directory=DIRECTORY
              move all SOURCE arguments into DIRECTORY

       -T, --no-target-directory
              treat DEST as a normal file

       -u, --update
              move only when the SOURCE file is newer than the
destination file or when the destination file is missing

       -v, --verbose
              explain what is being done

       --help display this help and exit

       --version
              output version information and exit
       The  backup  suffix is `~', unless set with --suffix or
SIMPLE_BACKUP_SUFFIX.  The version control method may be selected
via the --backup option or through
       the VERSION_CONTROL environment variable.  Here are the
values:

       none, off
              never make backups (even if --backup is given)

       numbered, t
              make numbered backups

       existing, nil
              numbered if numbered backups exist, simple otherwise

       simple, never
              always make simple backups

AUTHOR
       Written by Mike Parker, David MacKenzie, and Jim Meyering.

COPYRIGHT
       Copyright © 2010 Free Software Foundation, Inc. License
GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>.
```

```
NAME
       nona - Stitch a panorama image

SYNOPSIS
       nona [options] -o output project_file (image files)

DESCRIPTION
       nona uses the transform function from PanoTools, the
stitching itself is quite simple, no seam feathering is done.

       Only the non-antialiasing interpolators of PanoTools are
supported.

       The following output formats (n option of PanoTools p script
line) are supported:

       JPEG, TIFF, PNG  : Single image formats without feathered
blending
       TIFF_m          : multiple tiff files
       TIFF_multilayer : Multilayer tiff files, readable by The Gimp
2.0

OPTIONS
       General options:

       -c  Create coordinate images (only TIFF_m output)

       -v  Quiet, do not output progress indicators

       -t num
           Number of threads to be used (default: number of
available cores)

       The following options can be used to override settings in the
project file:

       -i num
           Remap only image with number num (can be specified
multiple times)

       -m str
           Set output file format (TIFF, TIFF_m, TIFF_multilayer,
EXR, EXR_m)

       -r ldr/hdr
           Set output mode:

           ldr - keep original bit depth and response
           hdr - merge to hdr
       -e exposure
           Set exposure for ldr mode

       -p TYPE
           Pixel type of the output. Can be one of:

           UINT8   8 bit unsigned integer
           UINT16  16 bit unsigned integer
           INT16   16 bit signed integer
           UINT32  32 bit unsigned integer
           INT32   32 bit signed integer
           FLOAT   32 bit floating point
       -z  Set compression type. Possible options for tiff output:

           NONE      no compression
           LZW       LZW compression
           DEFLATE   deflate compression

AUTHORS
       Written by Pablo d'Angelo. Also contains contributions from
Douglas Wilkins, Ippei Ukai, Ed Halley, Bruno Postle, Gerry
Patterson and Brent Townshend.

       This man page was written by Cyril Brulebois
<cyril.brulebois@enst-bretagne.fr> and is licensed under the same
terms as the hugin package itself.
```

```
NAME
       pano_modify - Change output parameters of project file

SYNOPSIS
       pano_modify [options] input.pto

DESCRIPTION
       pano_modify modifies a single Hugin .pto project.

OPTIONS
       -o, --output=file.pto
           Output Hugin PTO file. Default: <filename>_mod.pto

       -p, --projection=NUMBER
           Sets the output projection to number x

       --fov=AUTO|HFOV|HFOVxVFOV
           Sets field of view.

           AUTO: calculates optimal fov

           HFOV|HFOVxVFOV: set to given fov

       -s, --straighten
           Straightens the panorama

       -c, --center
           Centers the panorama

       --canvas=AUTO|WIDTHxHEIGHT
           Sets the output canvas size

           AUTO: calculate optimal canvas size

           WIDTHxHEIGHT: set to given size

       --crop=AUTO|left,right,top,bottom
           Sets the crop rectangle

           AUTO: autocrop panorama

           left,right,top,bottom: to given size

       -h, --help
           Shows help

AUTHORS
       Thomas Modes
```

```
NAME
       pto2mk - Generate hugin stitching Makefiles

SYNOPSIS
       pto2mk -o <output_makefile> -p <output_prefix> project_file

DESCRIPTION
       Hugin doesn't stitch panoramas, it creates a Makefile
containing all the rules to render, then blend the output image and
processes this Makefile with
       'make'.

       pto2mk will generate a platform specific Makefile from an
existing hugin .pto project.

OPTIONS
       -o file
           Output Makefile

       -p output_prefix
           Prefix of output panorama

AUTHORS
       Written by Pablo d'Angelo and others.
```

```
NAME
       rm - remove files or directories

SYNOPSIS
       rm [OPTION]... FILE...

DESCRIPTION
       This manual page documents the GNU version of rm.  rm removes
each specified file.  By default, it does not remove directories.

       If  the  -I or --interactive=once option is given, and there
are more than three files or the -r, -R, or --recursive are given,
then rm prompts the user for
       whether to proceed with the entire operation.  If the
response is not affirmative, the entire command is aborted.

       Otherwise, if a file is unwritable, standard input is a
terminal, and the -f or --force option is not given, or the -i  or
--interactive=always  option  is
       given, rm prompts the user for whether to remove the file.
If the response is not affirmative, the file is skipped.

OPTIONS
       Remove (unlink) the FILE(s).

       -f, --force
              ignore nonexistent files, never prompt

       -i     prompt before every removal

       -I     prompt  once before removing more than three files, or
when removing recursively.  Less intrusive than -i, while still
giving protection against most mistakes

       --interactive[=WHEN]
              prompt according to WHEN: never, once (-I), or always
(-i).  Without WHEN, prompt always

       --one-file-system
              when removing a hierarchy recursively, skip any
directory that is on a file system different from that of the
corresponding command line argument

       --no-preserve-root
              do not treat `/' specially

       --preserve-root
              do not remove `/' (default)

       -r, -R, --recursive
              remove directories and their contents recursively

       -v, --verbose
              explain what is being done

       --help display this help and exit
       --version
              output version information and exit

       By default, rm does not remove directories.  Use the --
recursive (-r or -R) option to remove each listed directory, too,
along with all of its contents.

AUTHOR
       Written by Paul Rubin, David MacKenzie, Richard M. Stallman,
and Jim Meyering.

COPYRIGHT
       Copyright © 2010 Free Software Foundation, Inc.  License
GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>.
       This is free software: you are free to change and
redistribute it.  There is NO WARRANTY, to the extent permitted by
law.
```

APPENDIX C. Project Files Examples

C.1 PROJECT FILE FOR AUTOOPTIMISER

```
# This document describes the script supported by the autooptimiser.
#
# Only lines starting with 'p', 'v', 'i', or 'm' are read.
# So you can add comments and info as you like by using
# other line starting characters.
# Do not start a line with !, it is used by adjust plugin and scripts.
# the * character indicated the end of the script file.
# The script must contain:
# one 'p'- line describing the output image (eg Panorama)
# one 'i'-line for each input image
# one or several 'v'- lines listing the variables to be optimized.
# the 'm'-line is optional and allows you to specify modes for the
#optimization.
# one 'c'-line for each pair of control points
# 'p'-line options
# w1000        width in pixels
# h600         height in pixels (default: width/2)
# f0            projection format,
#                  0 - rectilinear (for printing and viewing)
#                  1 - Cylindrical (for Printing and QTVR)
#                  2 - Equirectangular ( for Spherical panos), default
#                  3 - full-frame fisheye
# v360         horizontal field of view of panorama (default 360)
# nPICT        Panorama file format, one of:
#                  PICT          pict-file on macs, bmp-file on win
#(default)
#                  PSD           single layer Photoshop file, 48bits
#supported
#                  PNG           png-format, 48bits supported
#                  TIFF          tiff-format, 48bits supported
#                  PSD_mask      Photoshop file, one image per layer
#                                  + shape mask & feathered clip mask at
#overlap center
#                  PSD_nomask    Photoshop file, one image per layer,
#                  TIFF_mask     tiff-format, multi-file, one image per
#file, 48bit supported
#                                  alpha layer with feathered clip mask at
#overlap center
#                  TIFF_m        tiff-format, multi-file, one image per
#file, 48bit supported
#                                  alpha layer with non-feathered clip mask
#at image border
#                                  + shape mask & non-feathered clip mask
#at image border
#                  JPEG          Panoramic image in jpeg-format. Use with
#f1
#                                  for IBM Hotmedia panoramas.
#                  PAN           SmoothMove movie. Use only with f2.
#                  IVR           LivePicture IVR movie
#                                  cylindrical (format f1) or spherical
#(format f2)
#                  IVR_java      LivePicture Java Panorama,
#                                  cylindrical (format f1) or spherical
#(format f2)
```

```
#                  VRML          VRML background node, use only with f2 for
#panoramas, or
#                                  VRML-object for PTStereo
#                  QTVR          Apple QTVR-panomovie. Use only with f1
#                  3DMF          3DMF-object (PTStereo).
# n"QTVR w400 h300 c1"           additional viewer options in a quoted
#string together with format
#              the following options are recognized:
#                  w(width) and h(height) of viewer window (only QTVR on
#Macs)
#                  c(codec: 0-JPEG, 1-Cinepak, 2-Sorenson) (only QTVR on
#Macs)
#                  q(codec quality):
#                     0-high,1-normal,2-low    QTVR on Macs
#                     0-100(highest)           on other jpeg-formats (PAN,
#IVR, IVR_java, VRML)
#                  g  progressive jpeg (0-no, 1-yes) (PAN, IVR, IVR_java,
#VRML)
#                     Optimized JPEG (0-on(default), 2-disabled), (3-
#progressive with optimized disabled)
#                  p  initial pan angle ( QTVR on Macs, VRML, IVR)
#                  v  field of view (QTVR, VRML, IVR)
#                  Many more options can be set by editing the viewer
#scripts
# -buf         suppress buffer commands in the stitcher script generated by
#PTOptimizer.
#                  (buffer commands are now obsolete, -buf and +buf on i
#lines are now
#                  ignored when stitching) This option should be set if you
#wish
#                  to edit the final panoramic image, eg for the two PSD
#formats.
# a0.0 b1.0 c0.04  Options to create multiple images in PTInterpolate and
#PTMorpher.
#                      a denotes starting value,
#                      b end value
#                      c increment. 0 is left, 1 is right image.
#                      The above command interpolates/morphs two images and
#                      creates 25 intermediate frames.
# u10          width of feather for stitching all images. default:10
# k1           attempt color & brightness correction using image number as
#anchor
# b1           attempt brightness correction with no color change using
#image number as anchor
# d1           attempt color correction with no brightness change using
#image number as anchor
#              Do not use more than one of k, d, b.This is new method of
#correcting

p w800 nPSD_mask –buf

# The 'i' lines describe input images. One line per image is required
# unneeded paramiters for optimizing but needed for stitching can be
# set here and the optimizer will automaticaly add them to the o lines
# ----------------
# f0           projection format,
#                                 0 - rectilinear (normal lenses)
#                                 1 - Panoramic (Scanning cameras like
#Noblex)
#                                 2 - Circular fisheye
#                                 3 - full-frame fisheye
#                               4 - PSphere (equirectangular)
#                               8 - Orthographic. This is only allowed in
#PTStereo and
#                                      for the first image. This indicates
#a map or ground plan.
# w600          width in pixels
# h1000         height in pixels
# v82           horizontal field of view of image (required, but ignored for
#f8)
# y0            initial yaw angle (required)
# p43           initial pitch angle (required)
# r0            initial roll angle (required)
# a,b,c         initial lens correction coefficients (defaults a0 b0 c0,
#optional)
#                   (see
#http://www.panotools.org/dersch/barrel/barrel.html[*])
# d,e           initial lens offset in pixels(defaults d0 e0, optional).
#                   Used to correct for offset from center of image
#                   d - horizontal offset,
#                   e - vertical offset
# g,t           initial lens shear.  Use to remove slight misalignment
#                   of the line scanner relative to the film transport
#                   g - horizontal shear
#                   t - vertical shear
# u10           (obsolete, globally used on p line) specify width of feather
#for stitching. default:10
# S100,600,100,800   Selection(l,r,t,b), Only pixels inside the rectangle
#will be used for conversion.
#                        Original image size is used for all image
#parameters
#                        (e.g. field-of-view) refer to the original image.
# C100,600,100,800   Crop(l,r,t,b), Only pixels inside the rectangle will
#be used for conversion.
#                        Cropped image size is used for all image parameters
#                        (e.g. field-of-view) refer to the cropped part of
#the image.
# m20           (obsolete, use S & C) ignore a frame 20 pixels wide.
#default: 0
# mx100         (obsolete, use S & C) crop to brightest rectangle with size
#100x200;
# my200         (obsolete, use S & C) used only for circular fisheye images
#(f2)
# s0            (obsolete, ignored, always blend) specify placement of seam
#between buffer and image:
#                    0-middle of overlap('blend' ,default)
#                    1- at edge of image ('paste').
# o             (the small letter). Morph-to-fit using control points.
# k0            (obsolete, use p line correction still used with plugin)
#                   attempt color/brightness correction when merging image
#and buffer, one of:
#                   0 - no correction(default);
#                   1 - change image;
#                   2 - change buffer;
#                   3 - change both
#                   this feature does not work very well!
# X10           World coordinates of camera position, only used for PTStereo
# Y200          If the camera is aligned (yaw = pitch = roll = 0.0),
# Z-13.5        X is coordinate to the right, Y vertically up and
#               -Z is forward viewing direction.
# nName         Name of image (ignored by PTOptimizer used in PTStitcher)
#
```

```
#       Parameters in different images can be linked using '='
#       followed by the image number starting with 0.
#       Example 'v=0' sets horizontal field of view as in
#       image number 0. This feature works for the variables
#       v, a, b, c, (r, p, y with caution) d, e, g, and t
#

i f2 r0      p0    y0     v183    a0 b-0.1 c0  mx400 my400
i f2 r-0.5   p1    y182   v=0     a0 b-0.1 c0  mx400 my400

# 'v'-line options:
# -----------------
# Please note: the 'v'-line must come after the the 'i'-lines.
# Optimization variables are listed together with the image number
# starting at 0. There can be several v-lines.
#
# y0            Optimize yaw in image 0
# p1            Optimize pitch in image 1
# r2            Optimize roll in image 2
# v0            Optimize field of view in image 0
# a2            Optimize lens correction parameter 'a' in image 2
#                   b and c can be equally optimized.
# X1            Optimize x-coordinate of image 1, only for PTStereo
# Y2            Optimize y-coordinate of image 2, only for PTStereo
# Z6            Optimize z-coordinate of image 6, only for PTStereo
#
#       If a image has a parameter linked to another image only
#       need to optimize the master.

v v0 r0 p0 r1 p1 y1

# 'm'-line options
# ----------------
# Set mode for stitcher, not required
#
# g2.5          Set gamma value for internal computations (default 1.0)
#                   See #<http://www.panotools.org/dersch/gamma/gamma.html>
# i2            Set interpolator, See
#<http://www.panotools.org/dersch/interpolator/interpolator.html>
#                 one of:
#                    0 - poly3 (default)
#                    1 - spline16,
#                    2 - spline36,
#                    3 - sinc256,
#                    4 - spline64,
#                    5 - bilinear,
#                    6 - nearest neighbor,
#                    7 - sinc1024
#             \/ antialiasing filters \/ See
#<http://www.pano2qtvr.com/dll_patch/>
#                    8 - Box
#                    9 - Bartlett/Triangle
#                    10 - Hermite
#                    11 - Hanning
#                    12 - Hamming
#                    13 - Blackmann
#                    14 - Gaussian 1/sqrt(2)
#                    15 - Gaussian 1/2
#                    16 - Quadardic
#                    17 - Cubic
#                    18 - Catmull-Rom
#                    19 - Mitchell
#                    20 - Lanczos2
#                    21 - Lanczos3
#                    22 - Blackman/Bessel
#                    23 - Blackman/sinc
#
# p0           Create panorama after optimizing control points
#                  0 no(default), 1 yes

m g1.5 i6

#
# 'c' lines
# ----------------
# Control point lines
# One line per point pair
# about one pair of points per image per variable being optimized.
# The more variables being optimized the more control points needed.
#
# n0           first image
# N1           second image
# x1066.5       first image x point position
# y844.333      first image y point position
# X239.52       second image x point position
# Y804.64       second image y point position
# t0           type of control point (optional)
#                  0 - normal (default)
#                  1 - optimize horizontally only
#                  2 - optimize vertically only
#                  3+ (all other numbers) - straight line
#
# Every thing after # is ignored.
```

C.2 PROJECT FILE FOR NONA

```
# This document describes the script supported by the nona stitcher.
#
# Based on the PTStitcher documentation
#
#
# Only lines starting with 'p','o', i', 'm' or 'k' are read,
# so you can add comments and info as you like by using
# other line starting characters.

# The stitcher script must contain:
# one 'p'-line describing the output image (eg Panorama)
# one 'o'-line for each input image
# one 'm'-line for global options
#
# 'p'-line options
# w1000        width in pixels
# h600         height in pixels
# f0           projection format,
#                  0 - rectilinear (for printing and viewing)
#                  1 - Cylindrical (for Printing and QTVR)
#                  2 - Equirectangular ( for Spherical panos), default
#                  3 - full-frame fisheye
# v360         horizontal field of view of panorama (default 360)
# nPICT        Panorama file format, one of:
#                  PNG           png-format, 8 & 16 bit supported
#                  TIFF          tiff-format, all tiff types supported
#(8,16,32 bit int, float, double)
#                  TIFF_m        tiff-format, multi-file, one image per
#file
#                                  alpha layer with non-feathered clip mask
#3at image border
#                  TIFF_multilayer  tiff-format, multi-image-file, all
#files in one image
#                                  alpha layer with non-feathered clip mask
#at image border
#                                  This filetype is supported by The GIMP
#                  JPEG          Panoramic image in jpeg-format.
#               some more supported file formats (mostly only 8 bit
#support)
#                  PNM, PGM, BMP, SUN, VIFF
#
#              Special options for TIFF output:
#              n"TIFF c:NONE"
#                  c - select TIFF compression, possible options: NONE,
#LZW, DEFLATE
#
#              Special options for TIFF_m and TIFF_multilayer output:
#              n"TIFF c:NONE r:CROP"
#                  c - TIFF compression, possible options NONE, LZW,
#DEFLATE
#                  r - output only used image area (cropped output). The
#crop offsets
#                      are stored in the POSITIONX and POSITONY tiff tags
#                  p1 - save coordinate images (useful for further
#programs, like vignetting correction)
#
#              Special options for JPEG output:
#              n"JPEG q95"
#                  q - jpeg quality
#
# E12.3        exposure value for final panorama
# R1           stitching mode: 0: normal LDR mode, 1: HDR mode
# T"UINT8"     bitdepth of output images, possible values are
#               UINT8  -  8 bit unsigned
#               UINT16 - 16 bit unsigned
#               FLOAT  - 32 bit floating point
#              By default the bit depth of the input images is use.
#
# S100,600,100,800   Selection(left,right,top,bottom), Only pixels inside
#the rectangle
#                    will be rendered. Images that do not contain pixels in
#this area
#                    are not rendered/created.
#
# k1           Image number of reference image for photometric correction
#
# P"100 12"    Parameters for tuning projection, number of parameters
#depends on projection
#

p w1000 h600 f0 v360 E12.3

# The 'i' lines describe input images (nona also accepts 'o' line image
#descriptions)
#
# f0           projection format,
#                  0 - rectilinear (normal lenses)
#                  1 - Panoramic (Scanning cameras like Noblex)
#                  2 - Circular fisheye
#                  3 - full-frame fisheye
#                  4 - PSphere, equirectangular
# v82          horizontal field of view of image (required)
# y0           yaw angle (required)
# p43          pitch angle (required)
# r0           roll angle (required)
# a,b,c        lens correction coefficients (optional)
#                  (see http://www.fh-
#furtwangen.de/~dersch/barrel/barrel.html)
# d,e          initial lens offset in pixels(defaults d0 e0, optional).
#                  Used to correct for offset from center of image
#                  d - horizontal offset,
#                  e - vertical offset
# g,t          initial lens shear.  Use to remove slight misalignment
#                  of the line scanner relative to the film transport
#                  g - horizontal shear
#                  t - vertical shear
#
# Eev          exposure of image in EV (exposure values)
# Er           white balance factor for red channel
# Eb           white balance factor for blue channel
#
# Vm           vignetting correction mode (default 0):
#                  0: no vignetting correction
#                  1: radial vignetting correction (see j,k,l,o options)
#                  2: flatfield vignetting correction (see p option)
#                  4: proportional correction: i_new = i / corr.
#                       This mode is recommended for use with linear data.
#                       If the input data is gamma corrected, try adding
#g2.2
```

```
#                        to the m line.
#
#                       default is additive correction: i_new = i + corr
#                     Both radial and flatfield correction can be combined
#with the
#                      proportional correction by adding 4.
#                  Examples: i1 - radial polynomial correction by addition.
#                                  The coefficients j,k,l,o must be
#specified.
#                            i5 - radial polynomial correction by division.
#                                  The coefficients j,k,l,o must be
#specified.
#                            i6 - flatfield correction by division.
#                                  The flatfield image should be specified
#with the p option
#
# Va,Vb,Vc,Vd  vignetting correction coefficients. (defaults: 1,0,0,0)
#               ( 0, 2, 4, 6 order polynomial coefficients):
#                corr = ( i + j*r^2 + k*r^4 + l*r^6), where r is the
#distance from the image center
#              The corrected pixel value is calculated with: i_new = i_old
#+ corr
#              if additive correction is used (default)
#                         for proportional correction (h5): i_new = i_old /
#corr;
#
# Vx,Vy        radial vignetting correction offset in pixels (defaults Vx0
#Vy0, optional).
#                 Used to correct for offset from center of image
#                  Vx - horizontal offset
#                  Vy - vertical offset
#
# Vf           filename of flatfield image.
#               For additive correction the image will be used as it is.
#               In the case of correction by division, the flatfield will
#be divided by
#               its mean value.
#
# Ra,Rb,Rc,Rd,Re EMoR photometric model parameters. (defaults: 0,0,0,0,0)
#
# TrX,TrY,TrZ  mosaic mode translation offsets.
#
# S100,600,100,800   Selection(l,r,t,b), Only pixels inside the rectangle
#will be used for conversion.
#                        Original image size is used for all image
#parameters
#                        (e.g. field-of-view) refer to the original image.
#                        Selection can be outside image dimension.
#                        The selection will be circular for circular fisheye
#images, and
#                        rectangular for all other projection formats
#
# j0           stack number
#
# nName        file name of the input image.

i f2 r0   p0    y0     v183    a0 b-0.1 c0  S100,600,100,800 n"photo1.jpg"
i f2 r0   p0    y180   v183    a0 b-0.1 c0  S100,600,100,800 n"photo1.jpg"

# 'm'-line options
# ----------------
# Set mode for stitcher, not required
#
#
# g2.5         Set gamma value for internal computations (default 1.0)
#                  See <http://www.fh-
#furtwangen.de/~dersch/gamma/gamma.html>
#               This is especially useful in conjunction with the
#vignetting correction
#               by division
#
# i2           Set interpolator, See <http://www.fh-
#furtwangen.de/~dersch/interpolator/interpolator.html>
#                one of:
#                   0 - poly3 (default)
#                   1 - spline16,
#                   2 - spline36,
#                   3 - sinc256,
#                   4 - spline64,
#                   5 - bilinear,
#                   6 - nearest neighbor,
#                   7 - sinc1024
#

m i2

# 'v'-line options
# ----------------
# Indicate i-line parameters to optimise
# nona ignores all 'v' lines, these lines are used by autooptimiser
# (a,b,c,d,e,v,r,p,y geometric parameters) and vig_optimize
# (Eev,Er,Eb,Va,Vb,Vc,Vd,Vx,Vy,Ra,Rb,Rc,Rd,Re photometric parameters)
#
# The format is described in libpano13 Optimize.txt

# 'k'-line options
# ----------------
# Optional image masks are described by a 'k' line
#
#  i2           Set image number this mask applies to
#
#  t0           Type for mask:
#                   0 - negative (exclude region)
#                   1 - positive (include region)
#                   2 - negative, stack aware (exclude region from stack)
#                   3 - positive, stack aware (include region from stack)
#
#  p"1262 2159 1402 2065 1468 2003"  List of node coordinates
#               Coordinates are in pairs, at least three pairs are required

k i2 t0 p"1262 2159 1402 2065 1468 2003"

# 'c'-lines
# ----------------
# Control point lines
# nona ignores all 'c' lines, these lines are used by autooptimiser
#
# Every thing after # is ignored.
```