




**Universitat
Autònoma
de Barcelona**

Fast Computer Vision Algorithms applied to Motion Detection and Mosaicing

A dissertation submitted by **Marc Vivet Tañà** at Universitat Autònoma de Barcelona to fulfil the degree of **Doctor en Informàtica**.

Bellaterra, May 12, 2013

Director: **Dr. Xavier Binefa Valls**
Universitat Pompeu Fàbra
Dept. Tecnologies de la Informació i Comunicacions
Tutor: **Dr. Enric Martí Godià**
Universitat Autònoma de Barcelona
Dept. Ciències de la Computació



This document was typeset by the author using L^AT_EX 2_ε.

The research described in this book was carried out at the Computer Vision Center, Universitat Autònoma de Barcelona.

Copyright © 2013 by Marc Vivet Tañà. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the author.

ISBN 84-922529-8-7

Printed by Ediciones Gráficas Rey, S.L.

a la meva família

Acknowledgments

Primer de tot vull agrair al Xavier Binefa. Si no hagués estat per ell no hauria pensat en fer un doctorat i menys en visió. Per mi la Visió per Computador sempre la havia vist com aquelles assignatures tant difícils que no fas mai per por. Sort que el Xavier hem va adreçar a temps. Ara la visió per computació és una passió, un hobby i una feina al mateix temps. Per tot això i per tot el que m'has ensenyat aquests anys moltes gràcies.

I would especially like to thank Professor Shumel Peleg for welcoming me to his laboratory in Jerusalem and treating me as though was at home. I am deeply grateful to him for all that he taught me and for the patience that he had with my English. In addition, I would like to thank everyone met in Jerusalem.

També vull agrair especialment al Brais Martínez, que realment ha estat clau per dur a terme aquesta tesi. Es pot dir que ell ha estat el meu segon director. Vull agrair totes les hores que ha dedicat a ajudar-me i corregir els articles que he anat publicant, així com aquesta tesis.

A l'Enric Martí per acceptar ser el meu tutor i pels anys que hem col·laborat a les assignatures de Gràfics per Computador 2 i Bases de Dades.

Als meus companys de grup, que han fet que aquests quatre anys hagin estat fantàstics i que sempre més recordaré amb nostàlgia. Vull agrair al Ciro Gracia, qui va ser el meu primer company de despatx, i amb qui hem passat infinitats de anècdotes de tots tipus, des de estressants com algun que altre hackeig fins a experiències inoblidables com travessies amb Kayak. També vull agrair a l'Oriol Martínez, que es la persona amb qui he dinat més cops de restaurant i el culpable de passar de fanàtic de iOS a fanàtic de Android! Al Luís Ferraz amb qui m'ho he passat pipa anant de congressos, tot i que es dur compartir despatx amb ell ja que programa amb veu alta. Al Pol Cirujeda que hem va descobrir el món de les aplicacions per mòbils i també hem va introduir al món de la fotografia. Al Luís Ruiz que va aconseguir programar el DLIG amb C++ una fita de la que desitjo que s'hagi recuperat. Al Xavier Mateo que hem va descobrir el món de les PDAs i mil programets la mar d'útils. I la resta de persones que he coincidit com l'Adrià Ruiz, l'Adrià Pérez, el Ramon Lluís, el Rafa i el J.

Al secretariat de la UPF, la Vanessa, la Magda, la Santa, la Lydia, la Beatriz, la Jana, la Joana, Motse i la Judith. Que m'han ajudat moltíssim aquests anys tot i no ser de la UPF.

To my colleagues at Imagination Technologies and especially to my manager Paul Brasnett for letting me use the laboratory to finish this thesis and to James Brodie who made the last revision to this thesis.

A la gent de owl, Enric Vergara, Ivan Molera i Marçal Prats, que gracies a la seva dedicació i empenta vam crear dos aplicacions per iPhone i Android que més tard van derivar a un article.

A la meva família que no han parat de pressionar perquè acabés la tesis d'una vegada per totes. Als meus pares Jordi i Dolors; als meus germans David i Laura; als cunyats; Marçal i Xènia; als Nebots Lluc i Nil; i a la sogra Pilar.

I finalment vull agrair a la Raquel, la meva parella, tot el suport que m'ha donat tots aquests anys. I també per la paciència que ha tingut amb mi, ja sigui perquè havia de marxar 3 mesos o per haver de treballar els caps de setmana. També vull agrair-li la força que m'ha donat per no abandonar la tesi i recordar-me a diari que dediques unes hores a escriure. Per tot això moltes gràcies.

Abstract

Motion detection is widely used as the first step in many computer vision processes. Motion is usually related to interesting regions, for example cars, people, etc. Hence, it is a good criterion to use these regions as initialization for other, more computationally demanding processes such as classification and action recognition. This means that for the correct operation of the overall system, it is very important to obtain the best segmentation possible in this first step.

When the camera is not static, the motion produced by the camera hides the motion present in the scene. Therefore, it is necessary to compensate the camera motion. In other words, all frames must be aligned into a single coordinate system. As well as motion detection, by stitching all aligned frames we can generate a representation of the whole field of view from the camera. This process is called mosaicing and the result is the mosaic image.

This work is focused on motion detection and mosaicing which are fields of study commonly used in video surveillance applications, where processing in real-time processing is mandatory. Due to the restriction of computational time, we try to find a suitable compromise between performance and accuracy.

We start this thesis by presenting a methodology for improving motion detection on static cameras. The proposed method is based on combining the output of different motion detectors through a Markov Random Field framework. In addition, we take advantage of the nature of Markov Networks to add spatial and temporal consistency.

We then extend the previous method for the case of non-static cameras. The camera motion (caused by panning, tilting and zooming) is estimated using Multiple Kernel Tracking, which it is extremely efficient in terms of computation time. In addition, we show how to generate the mosaic image in real-time, with low memory requirements and robust to close path camera movements.

Continuing to the mosaic generation, we go one step further by presenting an alignment method that is efficient and also very precise. We call this method Direct Local Indirect Global (DLIG). The key idea of the DLIG alignment is to divide the frame alignment problem into the problem of registering a set of spatially related image patches. Patch-based registration allows us to prevent the most significant problems in mosaicing by performing multiframe registration, such as error accumulation or precise degeneration.

The last method proposed in this work is for mosaic generation when the camera has translational movement. In this case, frame alignment is even more difficult, since the presence of parallax makes it impossible to use the previous alignment methods. However, this presence of parallax allows us to generate mosaic images with a 3D effect for the viewer. We perform the alignment in a space-time volume, where the alignment is depth invariant. Furthermore, we present a novel stitching method to efficiently apply pyramid blending in video mosaics. The proposed method (“Barcode Blending”) overcomes the complexity of building pyramids for multiple narrow strips, combining all strips in a single blending step.

At the end of this work, we present the conclusions grouped by chapters, pointing out the strongest and most important aspects of the proposed methods and the direction of future work. Finally, we wish to remark that each chapter of this thesis is based on a published article.

Contents

1	Introduction	1
2	Background Subtraction for Static Cameras based on MRF	7
2.1	Introduction	7
2.2	Markov Random Fields	11
2.3	Belief Propagation	15
2.4	Our model	17
2.5	Experimental results	22
3	Background Subtraction for Non-static cameras based on MKT	27
3.1	Introduction	27
3.2	Kernel Tracking	29
3.2.1	Mean Shift Tracking	29
3.2.2	Multiple Kernel Tracking with SSD	31
3.3	Global Tracking Model and Maintenance	33
3.4	Experimental Results	37
4	DLIG: Direct Local Indirect Global alignment for Video Mosaicing	43
4.1	Introduction	43
4.2	Overview of the Proposed Method	46
4.2.1	Direct Local Indirect Global Alignment	46
4.2.2	Multi-frame Matching	46
4.3	Method Description	48
4.3.1	Local Point Correspondences using a Tracking Algorithm	49
4.3.2	Direct Local Indirect Global Alignment Computation	50
4.3.3	Mosaic Construction using the DLIG Algorithm	51
4.4	Description of the tracking algorithms considered in this work	54
4.4.1	Normalized Cross-Correlation	54
4.4.2	Lucas Kanade	55
	Lucas Kanade Additive	56
	Lucas Kanade Compositional	59
	Lucas Kanade Inverse Additive	60
	Lucas Kanade Inverse Compositional	61
	Feature method based on SIFT	61
4.5	Experimental Results	62

4.5.1	Description of the test videos	62
4.5.2	Implementation details and results obtained	64
4.5.3	Memory requirements and computational cost	72
5	Real-Time Stereo Mosaicing using Feature Tracking	77
5.1	Introduction	77
5.2	3D Motion Computation	80
5.2.1	Time Warping Motion Model	80
5.2.2	3D Alignment by Time Warping	82
5.3	Mosaic Construction	84
5.3.1	Multi-Band Blending	85
5.3.2	Barcode Blending	87
5.4	Experimental Results	89
6	Conclusions and Future Work	95
6.1	Background Subtraction for Static Cameras based on MRF	95
6.2	Background Subtraction for Non-static cameras based on MKT	96
6.3	DLIG: Direct Local Indirect Global alignment for Video Mosaicing	97
6.4	Real-Time Stereo Mosaicing using Feature Tracking	98
6.5	Future Work	98
	Bibliography	103

List of Tables

2.1	Codification of the different permutations of the detectors. <i>D.</i> means Detector.	19
4.1	Alignment error comparison. SR stands for SIFT + RANSAC and KT for Kernel Tracking. The graphics show in the <i>x</i> -axis the frame number, from 1 to 70, and on the <i>y</i> -axis (ranging from 0 to 20) the computed error in terms of mean for the frame pixels (red) and their quantized distribution (grey). See text for details on the graphic construction. Error/Frame <i>D.</i> means Error/Frame Distribution.	65
4.2	Alignment error comparison. SR stands for SIFT + RANSAC and KT for Kernel Tracking. The graphics show in the <i>x</i> -axis the frame number, from 1 to 70, and on the <i>y</i> -axis (ranging from 0 to 20) the computed error in terms of mean for the frame pixels (red) and their quantized distribution (grey). See text for details on the graphic construction. Error/Frame <i>D.</i> means Error/Frame Distribution.	66
4.3	DLIG alignment accuracy depending on the Kernels Tracking configuration on progressive videos	68
4.4	DLIG alignment accuracy depending on the Kernel Tracking configuration on interlaced videos	69
4.5	Kernel Tracking DLIG mosaicing accuracy under two illumination change patterns. I.V. stands for intensity variation. F stands for failed mosaics. The graphics are constructed as in table 4.1 (see text for further explanation) . . .	69
4.6	Kernel Tracking DLIG mosaicing accuracy under two illumination change patterns. I.V. stands for intensity variation. F stands for failed mosaics. The graphics are constructed like in table 4.1 (see text for further explanation) . .	70

List of Figures

1.1	Representation of a spatio-temporal associated to a static scene recorded from a static (a) and a vibrating camera (b). The red lines represents epipolar lines connecting the three tree tips present in the sequence. While the epipolar trajectories in (a) are completely straight, in (b) they are distorted by the vibration of the camera.	2
1.2	Epipolar lines when the recorded sequences include motion parallax under limited camera motion. (a) and (b) are two video sequences of a scene of a man (close to the camera) and the sun (far from the camera), represented in a time space volume. The bottom figures represent the $x - t$ epipolar plane of the sequence. The red and blue lines correspond to two epipolar lines defined by the trajectories of the man's hat and the center of the sun. The video sequence of (a) is recorded using a camera that has a constant translational movement perpendicular to its field of view, while (b) is recorded with a camera where the translational movement is not constant. In (a) the two epipolar lines are completely straight and the slope of these lines depends on the depth of the object. In the first figures of (b) the two epipolar lines are not straight, because of the variations in the camera speed. Time warping is used in the last two figures of (b) in order to straighten the red epipolar line without distorting the blue epipolar line. Note, that shifting the frames in (b) to straighten the red epipolar line distorts the blue epipolar line.	5
2.1	Pixel models of different background subtraction algorithms. In black is the histogram of a concrete pixel. In red is the estimated probability density function. (a) Threshold centered on the mean. (b) Gaussian distribution. (c) Mixture of Gaussians. (d) Kernel Density Estimator.	8
2.2	Relation between adjacent pixels. (a) Shows a frame of a video sequence, with a orange rectangle. (b) is the enlarged version of the pixels inside the orange rectangle. (c) shows a graphical model describing the relations between pixels.	10
2.3	Example of different graphical models. (a) is a directed graph, (b) is an undirected graph and (c) is an undirected factor graph. (b) and (c) are equivalent Markov Random Field networks, while (a) is a Bayesian network.	12
2.4	Example of <i>Markov property</i> . Node b has a fixed value, so that nodes a and b are mutually independent.	13

2.5	A <i>maximal clique</i> is marked in blue and a <i>clique</i> in red.	13
2.6	Messages involved in the belief propagation <i>Sum-Product</i> algorithm in a 1D graph with two single nodes. The blue arrows are the messages from nodes to factors and the red arrows the messages from factor to nodes.	16
2.7	MRF with a lattice topology, where each node is connected to the nodes placed in his left, right, up and down. This is a common topology for graphs in computer vision, as it can represent the pixels in an image and reproduce their spatial layout.	16
2.8	Different directions of the 4 connected Markov Random Field.	17
2.9	Bloc diagram of the proposed method. t indicates the frame index, BS is refers to Background Subtraction. SD refers to Shadow Detector and ED refers to Edge Detector	18
2.10	Graphical representations of the influence of the three different information sources on the graph. (a) Shows the direct representation, where each source is connected independently through three different functions. (b) The output of each binary source of information is codified to generate a single function of each state. This is the representation used in this section.	19
2.11	The left image corresponds to a representation of our model. Spheres represent the variable nodes belonging to X , cylinders represent the variable nodes corresponding to D , cubes represent factors from F and pyramids represent factors from H . The right image shows all the connections of one node. . . .	20
2.12	The left image shows the result of performing one iteration of BP. The center image shows the result of performing five iterations. The right image shows the difference between motion masks.	23
2.13	Motion segmentation using an IR sequence.	23
2.14	The top images are the results of the color background subtraction (left) and the edge detector (right). The bottom left shows the shadow detector result and on the right is the result after combining all previous outputs using the proposed method.	24
2.15	Comparison of different pixel-level background subtractions techniques, applied to the same video sequence. The first image is the original frame, the second is the difference or probability given by the algorithm, the third image is the output of the algorithm (motion mask) and the last image is the output improved by our method.	25
2.16	Comparison of Multi-modal background subtraction techniques at pixel-level, applied to the same video sequence. The first image is the original frame, the second is the difference or probability given by the algorithm, the third image is the output of the algorithm (motion mask) and the last image is the output improved by our method.	26
3.1	Target regions distributed in a uniform lattice over the frame. Target regions are represented by green circles. The frame margin is indicated by the dashed rectangle.	34

3.2 Video sequence recorded with a camera vibrating due to traffic. The right image (b) shows the result of modelling the background image directly and the left (a) image shows the result of modelling the background image using motion compensation. Each image is composed of 16 pieces, with the same size, corresponding to the first frame in the video sequence (F), the generated background model (M) and the difference between the first frame and the background model (D). It can be seen that (a) correctly generates the background model, while (b) generates a blurry background. 35

3.3 Example of target region configuration. The red box represents the area occupied by the current frame in the f_r space, the dashed box is the margin (only target regions that fit inside this box are selected), purple circles represent target regions of \mathbf{Q} that are not selected, green circles represent target regions selected to align the current frame (and they will be updated) and white circles represent target regions that added to \mathbf{Q} in this iteration. 36

3.4 Example of an indoor scene. The top image shows the background mosaic generated for this sequence. The bottom images show, from left to right: the original frame; the result obtained using a Gaussian background subtraction; the motion mask once we have applied the Markov Random Field background subtraction method proposed in the previous chapter; and the motion mask overlapped with the frame. 38

3.5 Example of an outdoor scene. The top image shows the background mosaic generated for this sequence. The bottom images show, from left to right: the original frame; the result obtained using a Gaussian background subtraction; the motion mask once we have applied the Markov Random Field background subtraction method proposed in the previous chapter; the motion mask overlapped with the frame. 38

3.6 Results obtained in a set of consecutive frames (indoor video sequence). Note that the camera movement can be appreciated by comparing the foreground of the first and the last frame in the sequence. 39

3.7 Results obtained in a set of consecutive frames (outdoor video sequence). Note that the camera movement can be appreciated by comparing the foreground of the first an the last frame in the sequence. 39

3.8 State of frame 923 in a long term video sequence. The top figure is the target regions and their locations over the mosaic image. The middle images are the Gaussian background mosaic, where the left image is the mean mosaic and the right image is the variance mosaic. Finally, the bottom images show, from left to right, the original frame, the motion mask obtained with the Gaussian background subtraction and the result after using our Markov Random Field background subtraction method proposed in the previous chapter. 40

3.9 State of frame 2471 in a long term video sequence. The top figure is the target regions and their locations over the mosaic image. The middle images are the Gaussian background mosaic, where the left image is the mean mosaic and the right image is the variance mosaic. Finally, the bottom images show, from left to right, the original frame, the motion mask obtained with the Gaussian background subtraction and the result after using our Markov Random Field background subtraction method proposed in the previous chapter. 41

4.1	Precision degeneration. Euclidean distance is equivalent to point matching error. The circles drawn around points x'_1 and x'_2 show points at equal distance to their central locations. The interframe distortion is almost non-existent. In contrast, when projected onto the mosaic, they can suffer a great distortion, which hinders frame-to-mosaic alignment.	48
4.2	Green squares are patches centered at \bar{X}_i^t . The intensity of the color reflects the weights used by the WMSE. Those regions over the moving car have consistently low weight and therefore have a low impact on the alignment computation. The upper images are 4 consecutive frames; the lower image represents the corresponding mosaic.	52
4.3	Histogram of error produced (Euclidean distance in pixels) with respect to the algorithm output and a ground truth. This graphic was made by tracking 8000 points in ten different synthetic images.	57
4.4	Histogram of iterations needed to converge for each Lucas Kanade-based algorithm. The results were obtained by tracking 8000 point in ten different synthetic images.	58
4.5	The top half shows a scheme of how the video sequences are generated (projective pattern on the left, affine one on the right). On the bottom, the images used to generate the 16 test sequences.	63
4.6	A progressive frame from a test sequence (left), and the same frame of the interlaced version of the video sequence (right).	67
4.7	Left images: Region representation using different number of kernels (1, 4 or 9). Right: a projective deformation of a region produces anisotropic kernels. It can be approximated using isotropic kernels	68
4.8	Comparison of light changes in a long interlaced sequence recorded using a Sony handycam DCR-PC110E. The blocks building the image correspond to frame 150, frame 37000, and their difference. It is important to note the color change, which produces some differences when subtracting the frames. . . .	71
4.9	The figure shows the mosaic generated with the entire sequence (2200 frames). The green line shows the camera's trajectory and the dots mark the point of view each 10 frames. The color code indicates the frame, where the yellow color is used for the first frames, and the red ones for the last. The comparisons result from subtracting frame 733 from 979, and from 1762 (two differences on the left) and 407 from 1223 and 1367 (the two on the right). . .	73
4.10	This example shows a 256-frame interlaced video, recorded from an UAV with a resolution of 720x576 pixels. A trapezoid is plotted every 10 frames showing the FOV, while the color of the trapezoid indicates time, where yellow marks the first frame and red the last one.	74
4.11	This example shows a 1000-frame progressive video, recorded using a hand-held photo camera (Cannon ixus 750) with a resolution of 640x480 pixels. A trapezoid is plotted every 10 frames showing the FOV, while the color of the trapezoid indicates time, where yellow marks the first frame and red the last one.	75

5.1 (a) Every region of the image sensor corresponds to a different viewing direction. (b) Stereo mosaics are generated by creating the left-eye mosaic from strips taken from the right side of the video frames, and the right-eye mosaic is created from strips taken from the left side of the video frames. (c) Stacking the video frames in an $x - y - t$ space-time volume. Creating a stereo mosaic can be seen as “slicing” the space-time volume by two $y - t$ planes. One on the left side and one on the right side. 79

5.2 Resulting epilines when the camera has constant motion and when the camera has variations in its motion. In figure (a) an example of a sequence recorded camera that has constant motion is shown. While in figure (b) the same sequence is shown, but in this case the camera has variations in its motion. By resampling the t , Time Warping is able to straighten the epilines. 79

5.3 Scheme of how tracking a target can describe an EPI line in the $x - y - t$ space-time volume. The red line represents an EPI line, and the rectangles represents frames of a video sequence panning from left to right. The red circle represents the target. 81

5.4 Evolution of some target regions in a video sequence. Each circle represents a target and the color its state. From green to red the SSD error (green less error) is represented. The targets detected on flat regions are pink, targets out of bounds are black and new targets white. 83

5.5 Comparison between three different blending functions. The top images show the results after applying the transition functions that are shown in the bottom figures, where the red line is the function applied to the left image and the blue line is the function applied to the right image. (a) shows the results of applying a transition function with high slope. As can be seen, the edge between the two images is visible. (b) is the result of stitching the same images using a transition function with lower slope. Note that in this case, the edge between the two images is no longer visible. However, the image becomes more blurry (see the rocks in the lower part of the image). Finally, in figure (c) shows the result after applying Multi-Band Blending. In this case it is not possible to detect that this image is formed from two different images. The blending function, in this case, is composed of multiple transition functions, one for each pyramid level. High image frequencies are stitched together using high slopes, while lower image frequencies are stitched together using lower slopes. 86

5.6 Example of Gaussian Pyramid and Laplacian pyramid. The first column of images (Gaussian pyramid) are subtracted to the second column (Expanded Gaussian pyramid) giving as a result the third column (the Laplacian pyramid). 88

5.7 (a) Mosaic construction as slicing the $x - y - t$ space-time volume. Five frames of a video sequence are shown in an aligned space-time volume. The generated mosaic is the blue plane P_{y-t} . Each frame is represented by a different color. (b) Image strips around the intersection of P , with each image are pasted onto P . In order to simplify the scheme, we assume that the slope of the epipolar lines $\mu_m = 1$. (c) Odd, even and mask mosaics generation, in order to compute Barcode Blending. 89

5.8	Example of odd, even and mask mosaic and the resultant mosaic when Barcode Blending is applied.	90
5.9	Comparison between mosaic regions with and without Barcode Blending. The top images belong to a mosaic image generated without any blending and the bottom images are generated using Barcode Blending.	91
5.10	An anaglyph stereo mosaic generated from a video sequence of 450 frames captured from a river boat. The bottom images show the same regions as they appear in the different mosaics, showing different points of view generated using the same aligned volume.	92
5.11	Anaglyph stereo mosaics generated from videos recorded by a hand held iPhone 4, with resolution of 1280×720 pixels.	93
5.12	Snapily3D iPhone application developed using the technology presented in this chapter.	94

Chapter 1

Introduction

Detecting motion of objects in a video sequence is the first step in many computer vision applications, allowing for more complex processes. Regions with motion are often the interest regions where actions takes place, for example a person walking or a moving car. For this reason, those regions can be used, for example, as initialization for tracking algorithms, to reduce the processing area of an image for classification algorithms, etc. Therefore, obtaining a good segmentation for the proper operation of the overall process becomes an important step.

In the literature, regions with motion are often called foreground and regions without motion are called background. Using the simile of a play, the set is the background, which does not change during the whole performance, and the actors are the foreground. If we know how the background looks, we can approach the problem by searching those regions that differ from it. Using regular cameras (RGB or grayscale) the most common approaches to tackle this problem are those methods based on background subtraction. Background subtraction algorithms are based on subtracting pixel values between a frame and a representation of the background, usually called the background image. Regions with a high difference, under some measure, are considered to be the foreground. Approaching the problem in this way, the main challenge becomes generating a descriptive background image that encompasses the possible aspect changes the background might undergo (e.g. due to changing lighting conditions) while effectively discriminating the foreground.

There are multiple algorithms to obtain a background image, such as directly using a previous frame in the video sequence [24, 61, 79] or creating a generative model for the intensity/rgb values of each pixel through a mixture of Gaussian [78, 46, 37]. However, all background subtraction techniques assume that the video sequence is recorded using a completely static camera. If we represent the video sequence as a space-time volume, image features produce straight lines along the temporal axis. This situation is depicted in Fig. 1.1a, which represents a sequence observed from a static point of view, so for example the three tree tips form straight lines in the space-time volume. In particular, if we slice the space-time volume in the $x - t$ plane, the resultant image will consist of a set of vertical straight lines, describing the pixel values over the time of the y th row (the cut position). These images are

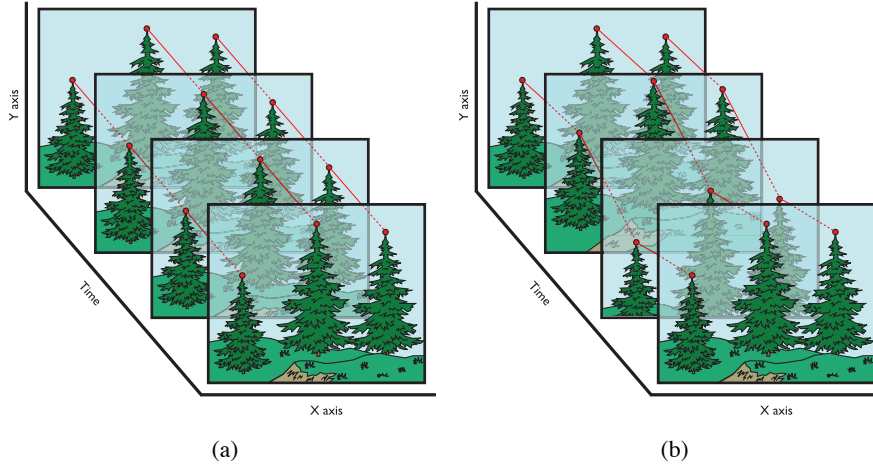


Figure 1.1: Representation of a spatio-temporal associated to a static scene recorded from a static (a) and a vibrating camera (b). The red lines represents epipolar lines connecting the three tree tips present in the sequence. While the epipolar trajectories in (a) are completely straight, in (b) they are distorted by the vibration of the camera.

called epipolar planes or EPI planes, and the straight lines along the temporal axis are called epipolar lines. Therefore, the methodology used by the majority of background subtraction algorithms to generate the background image can be seen as the study of the most common pixel value over the epipolar lines defined by each pixel location.

Imagine the case of a camera vibrating due to strong wind or heavy vehicle traffic. In such cases, there exists a 2D movement across frames that results in meaningless variations in the pixel values over time. Fig. 1.1b illustrates the consequences of this 2D movement in the space-time volume, where the same sequence is represented, but in this case the camera is not completely static. In this case, the points (the tree tips) are not aligned. Therefore, the straight lines defined by each pixel location in the space-time volume do not correspond to specific regions in the physical scene, causing erroneous background images and consequently poor motion segmentation. Therefore, before applying a background subtraction algorithm, it is necessary to first restore the relation between pixel locations and regions in the scene. In other words, we have to align the frames to convert the trajectories defined by specific points of the scene to straight lines.

There are two main groups of alignment methods, direct or featureless methods [88], and indirect or feature-based methods [13, 85]. Direct methods are those which use all the available frame pixels to compute an image-based error, which is then minimized to compute the alignment. In contrast, indirect methods compute the alignment by using a discrete set of frame locations. The point locations are selected to have representative local texture properties so that they are likely to be detected in corresponding image regions independently of the camera movement. Finally, the transformation is computed to minimize the error in the matching between the two sets of locations.

To align all frames in a video sequence it is necessary to select a reference frame so that its coordinates are used to dispose the rest of the frames. Usually the reference frame is the first frame or, to avoid aligning using foreground regions, the background image. This process is usually called motion compensation, because it is equivalent to compensating for the motion produced by the camera.

In many cases it can be necessary to detect motion over a wide area. Using a wide field-of-view (fish-eye) lens can be a partial solution, but these lenses cause large distortions in the captured video sequence. In addition, image quality and detail are compromised when a large area is captured. Another possible solution is to place multiple cameras in order to cover the entire area. However, increasing the number of cameras increases the costs of the system. Furthermore, in order to visualize the scene their fields-of-view have to be placed in correspondence. For these reasons, it is very common to use non-static cameras. For instance, robotic cameras like Pan-Tilt-Zoom cameras (PTZ), which are capable of rotating over a fixed point and zooming are commonly used.

Again, if we want to detect motion in the captured scene, we have to compensate the motion caused by the camera. However, in this case, aligning all frames with respect to a single reference frame might not be enough, as the reference frame and the candidate frame need to have a significant overlap in their field of view. For example, in the case of a camera rotating 360° using the first frame as a reference frame will only be possible for the initial frames. In this case, it is necessary to generate a representation of the whole captured area by aligning and stitching all frames into a single image (called the mosaic image). This process is called Mosaicing. Given an input from a moving camera, mosaicing allows, in combination with a background subtraction algorithm, to generate a background image for the whole scene. It is then possible to perform motion detection by putting a new frame in correspondence with a subregion of the background mosaic, and using any background subtraction technique that is used for static cameras.

Apart from motion detection, video mosaicing also gives valuable information to a surveillance system, such as the global localization of the current field of view within the global scene covered by the camera. With video mosaicing the effectiveness of the surveillance system is improved by presenting the information extracted by the camera to the operator in a more natural form. This allows, for example, to integrate a summary of the events observed by a camera over a time lapse in a single scene representation [39, 70, 66], or to perform spatial measurements that can be useful to automatically analyze behaviour, e.g. finding suspicious trajectories of pedestrians along the scene [57] or for automatically describing the most important actions in a soccer match [94]. In addition, video mosaicing can be used for generating high resolution images [77, 54, 60], representations of terrains [30, 29, 59], or video compression [40, 47] among other applications. Due to these reasons, construction of robust and efficient mosaics under general conditions is a relevant field of study.

Video mosaicing becomes challenging in the presence of motion parallax. Motion parallax is a depth cue that results from the camera motion. For example, in the case of a scene recorded from a moving vehicle, objects that are close to the camera move faster across the field of view than those that are further. In such a case, 2D image alignment methods cannot handle 3D parallax, producing distorted mosaics. Computing the 3D camera motion in the scene (egomotion), simplifies the mosaicing process, but it is a challenging problem when

only using image analysis. In addition, the high number of degrees of freedom of the camera makes the process very sensitive to errors. However, under limited camera motion, for example when the camera has a constant translational movement perpendicular to its field of view, or in other words, the camera has a constant velocity over a straight line parallel to the scene in the x direction. It is observable that epipolar lines are straight, but instead of being vertical, in this case, epipolar lines' slope depend on the distance from the camera (the depth). The farther the object is respect to the camera, the steeper the epipolar line is. Fig. 1.2a shows an example of a video sequence of a man and a sun recorded using a camera with constant translational movement perpendicular to its field of view. The sun is farther from the camera and the epipolar line described by its center is steeper than the one defined by the man. In this case, the mosaic image can be generated by slicing the space-time volume in the $y-t$ axis. Note that this procedure lets us generate different mosaic images depending on the x , where we slice the volume. The different x positions correspond to specific regions on the camera sensor, where the light arrives with a specific angle. Therefore, the different $y-t$ slices correspond to different view points of the scene. Then it is possible to generate 3D stereo mosaics by selecting two $y-t$ slices in correspondence with the left and right eyes.

When the camera motion is not constant, which is the most common case, epipolar lines are no longer straight and the mosaics become distorted. Therefore, it is necessary to straighten all epipolar lines in order to simulate constant camera motion. Shifting the frames in the x or y axes is not enough, they are depth variant and the video sequence includes motion parallax. However, the time-space representation can provide a depth invariant image alignment by resampling the t axis. This process is called *time warping*. An example of this can be seen in Fig. 1.2b, where the video sequence of a man and a sun is recorded using a camera with non-constant camera motion. In this case, the epipolar line described by the man's hat is not straight. However by resampling the t axis, the epipolar line is straightened and at the same time the epipolar line corresponding to the sun is not distorted. Note that by shifting the frames in the x or y axis so that the man's hat epipolar line is straight would distort the sun's epipolar line.

We have organized this thesis in six chapters where this introduction is the first one. In chapter 2 we discuss the problem of detecting motion using static cameras. To this end, an extensive description of the main background subtraction algorithms in the literature is presented. Then we present our own background subtraction algorithm, which combines different visual cues and uses a probabilistic graphical model to provide spatio-temporal consistency to the background model. Our model represents each pixel as a random variable with two states, background and foreground. Then, Markov Random Fields (MRF) is used to describe the correlation between neighbouring pixels in the space-time volume. In addition, we provide a general framework to combine different motion related information sources, in order to increase the accuracy of the motion mask.

The next step is to target at the problem of detecting motion when the camera is not static, which is dealt with at chapter 3. In particular, the case with no parallax is considered. This is a common case as PTZ cameras or aerial perspectives do not produce motion parallax. We propose to compensate for 2D affine transformations caused by the camera by using Multiple Kernel Tracking, assuming that the major part of the frame belongs to the background. We first introduce Multiple Kernel Tracking describing how it can be formulated for this particu-

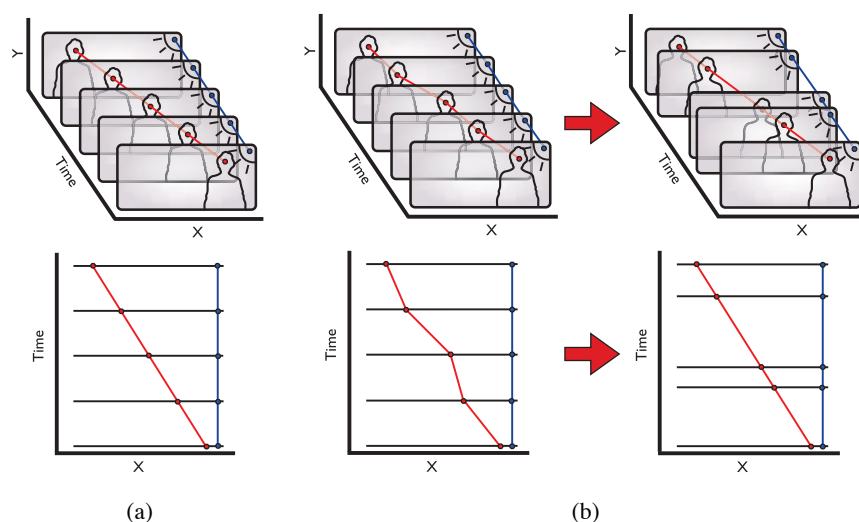


Figure 1.2: Epipolar lines when the recorded sequences include motion parallax under limited camera motion. (a) and (b) are two video sequences of a scene of a man (close to the camera) and the sun (far from the camera), represented in a time space volume. The bottom figures represent the $x-t$ epipolar plane of the sequence. The red and blue lines correspond to two epipolar lines defined by the trajectories of the man's hat and the center of the sun. The video sequence of (a) is recorded using a camera that has a constant translational movement perpendicular to its field of view, while (b) is recorded with a camera where the translational movement is not constant. In (a) the two epipolar lines are completely straight and the slope of these lines depends on the depth of the object. In the first figures of (b) the two epipolar lines are not straight, because of the variations in the camera speed. Time warping is used in the last two figures of (b) in order to straighten the red epipolar line without distorting the blue epipolar line. Note, that shifting the frames in (b) to straighten the red epipolar line distorts the blue epipolar line.

lar purposes. Then the generation of the background mosaic is defined so it is adaptable over time.

Chapter 4 presents a new frame alignment algorithm, the Direct Local Indirect Global (DLIG), which compensates the 2D motion using a projective transformation. The key idea of the DLIG alignment is to divide the frame alignment problem into the problem of registering a set of spatially related image patches. The registration is iteratively computed by sequentially imposing a good local match and global spatial coherence. The patch registration is performed using a tracking algorithm, so a very efficient local matching can be achieved. We use the patch-based registration to obtain multiframe registration, using the mosaic coordinates to relate the current frame to patches from different frames that partially share the current field of view. Multiframe registration prevents the error accumulation problem, one of the most important problems in mosaicing. We also show how to embed a Kernel Tracking algorithm in order to obtain a precise and efficient mosaicing algorithm.

In chapter 5 we face the problem of generating mosaics when the recorded scene contains motion parallax. We propose to align the video sequence in a space-time volume based on efficient feature tracking, for what Kernel Tracking is used. Computation is fast as the motion is computed only for a few regions of the image, yet still gives accurate 3D motion. This computation is faster and more accurate than the previous work that is based on a direct alignment method. We also present *Barcode Blending*, a new approach for using pyramid blending in video mosaics, which is very efficient. Barcode Blending overcomes the complexity of building pyramids for multiple narrow strips, combining all strips in a single blending step.

This thesis finishes with the conclusions and future work in chapter 6. We point out the strongest and most important aspects of our work, and some of the most promising lines of continuation of this work.

Chapter 2

Background Subtraction for Static Cameras based on MRF

In this chapter we consider the problem of detecting motion using static cameras. When considering this problem at pixel-level, it involves classifying each pixel within a video sequence into one of two classes: pixels belonging to moving regions (i.e. foreground pixels) or pixels which belongs to static regions (i.e. background pixels). Such classification can be performed using different cues, like pixel colour or scene edges, amongst others. However, it is possible to constrain the problem by noting that neighbouring pixels tend to have the same class, specially if no edges are present.

The main idea of this chapter is to combine different motion-related data and, together with the spatial and temporal consistency between pixels, improve the motion detection accuracy. We use a *Markov Random Field* (MRF) to build such a model, where each pixel is defined as a node connected to its spatio-temporal neighbours. Finally, we optimise the model by using a *Belief Propagation* algorithm, which has shown good convergence properties even in the case of loopy graphs such as the one used in this chapter.

2.1 Introduction

This chapter is dedicated to motion segmentation using background subtraction. Regions with motion are important because they are highly related to objects of interest. Using the same example as in the introduction, consider the case of a person walking or a moving car it is clear that these regions are more interesting than the asphalt or a wall. As its name indicates, background subtraction algorithms subtract/compare a given frame with a “background frame”, usually called the background image. The background image is intended to be a representation of the recorded scene, which does not contain regions with motion. Then, those regions with large differences (under some criteria) between the frame and the background images indicate where the regions with motion are located. The final out-

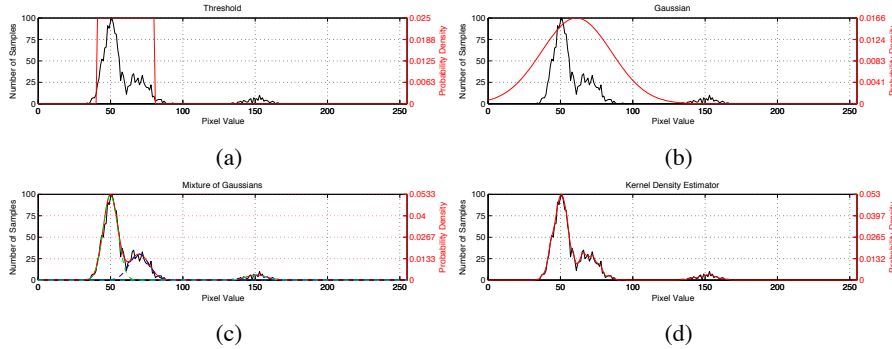


Figure 2.1: Pixel models of different background subtraction algorithms. In black is the histogram of a concrete pixel. In red is the estimated probability density function. (a) Threshold centered on the mean. (b) Gaussian distribution. (c) Mixture of Gaussians. (d) Kernel Density Estimator.

put is a binary mask (motion mask) indicating, for each pixel, if it is part of the foreground or the background. Note that it is assumed that the video sequence is recorded using a completely static camera, otherwise the motion caused by the camera can not be distinguished from the motion present in the scene.

There exists a large number of background subtraction techniques, which can be divided into two main groups: Pixel-level and Frame-level methods. Pixel-level methods are based on modelling each pixel independently in order to build a generative model of the background image. Modelling each pixel independently can simplify the background construction and speed up the process, but at the same time results in noisier motion masks. Therefore these techniques are adequate when efficiency is needed and higher noise levels are acceptable.

The most naive pixel-level method is the inter-frame difference [24, 61, 79]. The background image is set as the previous frame of the video sequence. Then the motion mask is set as those locations where the pixel difference is above a predefined threshold. The results of inter-frame difference depends on the object structure and speed, the camera frame rate and a global threshold, which makes this approach useless in most cases. Other basic methods consist of generating the background model depending on some statistics of the possible pixel values; for example, the average or the median of each pixel for the last n frames [50, 22]. These methods are very fast and provide an improvement over the frame-difference approach, although they still provide rather poor performance, and the storage of the previous n frames increases the memory requirements.

Memory requirements for the average and median background images can be met by computing the background image as the running average [15, 92] and the approximate median algorithm [56, 62]. When using such methods, the memory requirements are reduced from n frames to only one frame. A necessary addition to this method is to use the running average selectively. That is to say, a background image pixel will be updated only if the cor-

responding pixel of the current frame was classified as part of the background. Note that the methods described so far depend on a threshold applied for all frame pixels without distinction and that an explicit methodology to find the ideal threshold is not proposed, making it difficult to set up the system. In Fig. 2.1a an example of the probability density function of this kind of methods is shown.

To overcome the drawbacks of the previous methods, Wren et al. [89, 90], proposed a generative approach to model the possible value of each pixel associated with the background as a Gaussian distribution (see Fig. 2.1b). A heuristic can now be defined to select a good thresholding value by just considering the Mahalanobis distance associated with each pixel's distribution.

Similar approaches were presented in [78, 46, 37], among others. In these articles, the authors model each pixel by using a mixture of Gaussians, see Fig. 2.1c. Using a potentially multimodal distribution allows the generative model to adapt to shadows, specularities, swaying branches, computer monitors, among other equivalent real-world situations that cannot be modelled by using a single Gaussian. In order to speed up the PDF estimation, it uses a K-means approximation instead of an Expectation Maximization algorithm. However, in this approach the number of Gaussian distributions is arbitrarily pre-defined, the construction of the PDF becomes difficult, and also requires substantial computational time.

In order to simplify the PDF construction and maintain the multi-modality, [25, 96] proposes to model the background image using Kernel Density Estimation (KDE). KDE generates the PDF non-parametrically, so for each of the n most recent pixel intensity values become the mean of a Gaussian kernel of pre-defined covariance(see Fig. 2.1d). This procedure decreases the computational requirements with respect to the GMMs. However, the memory requirements are increased and the method is extremely sensitive to the kernel bandwidth, leading in many practical cases to a poor estimate of the true PDF. An alternative to the use of GMMs was presented in [34], where the authors use mean-shift to find the mode of the PDF. This method is closely related to the previous one, as the Mean Shift method is equivalent to computing the Maximum a Posteriori (MAP) of the PDF resulting from the KDE. However, the full PDF does not need to be computed.

[72, 42] proposed a method to impose temporal consistency on the classification of each pixel as background or foreground by using a Hidden Markov Model to model the transitions between the two states. This is still a pixel-level method, as only the values of a given pixel throughout the sequence are considered.

One common drawback of all pixel-level approaches is the lack of spatial consistency. That is to say, they neglect the fact that the neighbours of pixels belonging to the background are more likely to also be part of the background, and the same can be said about foreground pixels (see Fig. 2.2b). There is a relation between each pixel and its neighbours that can be used to improve the segmentation. Frame-level techniques are those that also use the spatial information in order to improve the final segmentation.

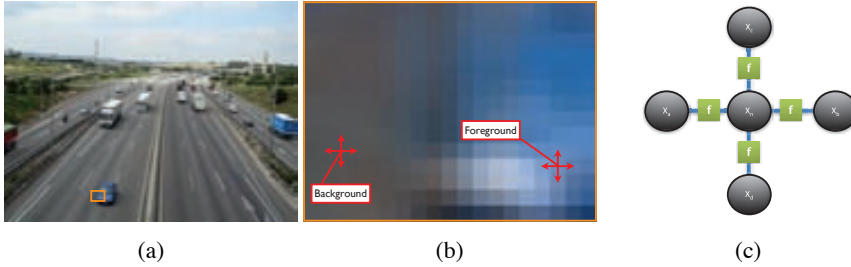


Figure 2.2: Relation between adjacent pixels. (a) Shows a frame of a video sequence, with a orange rectangle. (b) is the enlarged version of the pixels inside the orange rectangle. (c) shows a graphical model describing the relations between pixels.

The first method exploiting the spatial relation between pixels was the Wallflower background subtraction, presented in [82]. This method processes images at various spatial levels, pixel-level, region level and on the whole frame (frame-level), in order to have a better approximation of the background. It uses pixel-level components to make probabilistic predictions of the expected background using a linear filter based on the recent history of pixel intensity values. The region-level components fill in homogeneous regions of foreground objects, and the frame-level component detects sudden, global changes in the image in order to choose the best approximation of the background.

The eigen-background [63, 73, 49], is another frame-level technique. It uses principle component analysis (PCA) to model the background image. It generates a linear subspace by using the last n frames. In order to reduce the dimensionality of the space and to reduce the impact of noise, only the largest eigenvectors are kept. Note that moving objects do not have a significant contribution to the model because they do not appear in the same location during the n frames, whereas the background image can be accurately described as the sum of various eigenvectors. The motion mask is then found by projecting the frame onto the eigenspace and backprojecting. The regions which are not well described after backprojecting the image are assigned to the foreground.

Finally, the last group of background subtraction methods reviewed in this work are those methods based on graph models. More specifically, a Markov Random Field (MRF) is applied in these cases, as it perfectly matches the indirect nature of the relations between pixels, and the Markov property allows for efficient inference. In these methods, the spatial and in some cases temporal information is added to the output of pixel-level methods. For instance, [93] uses an MRF to introduce spatial and temporal information to frame differencing and [84] applies it for enforcing spatial consistency into the HMM-based method presented in [72]. By using MRFs, spatial consistency can be modelled by assigning a random variable with two states (background and foreground) to each pixel in the image, and connecting the neighbouring variables through edges that contain statistical models of the relation with its neighbours. This is depicted in Fig. 2.2c.

Once the relations between pixels are modelled, inference over the network has to be performed so that the optimal state for each node is obtained. Different techniques exist to perform the inference, such as Graph Cuts [44, 17] or Belief Propagation (BP) [91, 86]. The first one finds the best approximation of the optimum MRF state by repeatedly maximizing the joint probability using Max-flow / min-cut method. BP instead interactively propagates the probability (belief) of each node to its neighbours.

The work presented in this chapter makes use of a MRF. In particular, such a model is used to combine different cues extracted from the image and, at the same time, adding spatial and temporal consistency. This results in an improved motion segmentation. The different sources of information proposed in this work are a pixel-level RGB-based cue, a shadow detector, and an edge detector. The pixel RGB cue is obtained by fitting a 3-dimensional Gaussian distribution to model the possible RGB values for each pixel location. Then for a new pixel value corresponding to the new frame a thresholding is applied over the Mahalanobis distance yielded by the Gaussian distribution, and a value is assigned to the binary label. The shadow detector is an implementation of the Sakbot system [21, 22], which uses the background image given by the previous source as a basis to compare the variations in the luminance and chrominance over the HSV color space. Finally, the edge detector is computed as the result of applying the Sobel edge detector to the current image and then removing the edges present in the background image. Note that all the sources are binary. All of these methods constitute a current research direction and none of the solutions adopted in this chapter are optimal. Instead, the research interest is on how to combine different and complementary sources of information within a framework that also provide spatial and temporal consistency. Each source of information can be considered an independent module and could be replaced by a better algorithm, given that the output is comparable. Therefore, what is important is to highlight the remarkable improvement obtained when using the fusion of all modules with respect to the results obtained when using each method on its own.

The remainder of the chapter is as follows: we start by explaining in detail Markov Random Fields and Belief Propagation (Sec. 2.2 and 2.3); then section 2.4 presents the proposed background subtraction method; finally, section 2.5 describes the experimental results.

2.2 Markov Random Fields

Probabilistic graphical models are diagrammatic representations of probability distributions used to provide a simple way to visualize the structure, provide insight into the properties and abstract complex computations of probabilistic models. A graphical model is composed of nodes, representing random variables, and edges, representing the probabilistic relation between these variables [10, 11].

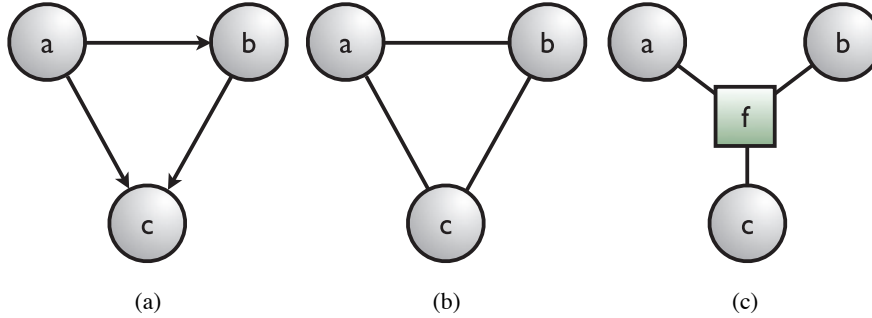


Figure 2.3: Example of different graphical models. (a) is a directed graph, (b) is an undirected graph and (c) is an undirected factor graph. (b) and (c) are equivalent Markov Random Field networks, while (a) is a Bayesian network.

Graphical models can be divided into two main classes depending on the nature of the edges. First of all, those graphical models represented by directed graphs also called **Bayesian networks**. In this class, the links between nodes have a particular directionality indicated by arrows. This kind of graph is useful for expressing causal relationships between random variables. For example, the equation:

$$p(a, b, c) = p(c|a, b)p(b|a)p(a) \quad (2.1)$$

is the joint probability of the direct graph shown in Fig. 2.3a, where nodes a , b and c are related using arrows. Node c is pointed to by nodes a and b , while at the same time, node a is pointing towards node b . Note that node a is not pointed at by any node and therefore its probability is not conditioned to any other node.

The other major class is that of undirected graphs, which is the class of **Markov Random Fields**. MRFs are graphical models where the edges do not have a direction and, therefore, the relations are bidirectional. Undirected graphical models are better suited to expressing soft constraints between random variables that do not correspond to causality. The visual equivalent of the previous example (Fig. 2.3a) can be seen in Fig. 2.3b. In this case, the relations do not have causal effect and therefore, the joint probability can not be defined in terms of conditional probabilities. Now it is possible instead to write the joint probability in terms of factors:

$$p(a, b, c) = \frac{1}{Z} \psi_1(a, b) \psi_2(a, c) \psi_3(b, c) \quad (2.2)$$

where Z is a normalization constant.

In this work, the edges encode the probabilistic relation between pixels. In such a case, relationships are not causal, as the influence between related pixels goes in both directions. Therefore, undirected graphs are the appropriate graphs to represent such relations.

The main characteristic of MRFs, apart from the indirect nature of their edges, is that they exploit the *Markov property*. This means that nodes are only affected by their neighbouring nodes. That is to say, a node is independent from the rest of the graph once the value of its neighbouring nodes is set. An example is shown in Fig. 2.4, where a fixed value is assigned to node b of the graph, implying that $p(a|c) = p(a)$ and $p(c|a) = p(c)$. The Markov property allows the decomposition of undirected graphs in terms of factors, each of which depends on a reduced number of variables defined by what is called a *clique*.

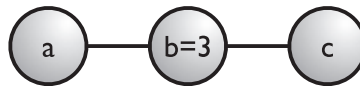


Figure 2.4: Example of *Markov property*. Node b has a fixed value, so that nodes a and b are mutually independent.

A *clique* is defined as a subset of nodes in a graph such that there exists a link between all pairs of nodes within it [10]. In other words, it is a subset of nodes where all of the nodes are fully connected. A *maximal clique* is defined as a clique such that it is not possible to include any other node from the graph on the subset and still be a clique. An example of a clique and a maximal clique can be found in Fig. 2.5, where a fully connected Markov Random Field with four nodes is represented. In this example, the maximal clique is depicted in blue, while another (non-maximal) clique is shown in red. Note that in this graph any combination of nodes defines a clique.

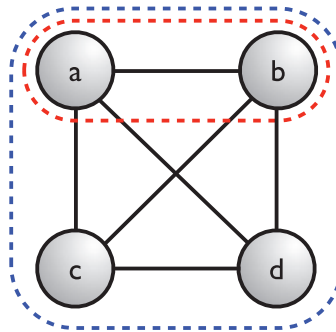


Figure 2.5: A *maximal clique* is marked in blue and a *clique* in red.

If we denote a clique as C , then the join probability in a MRF is defined as:

$$p(x) = \frac{1}{Z} \prod_C \psi_C(x_C) \quad (2.3)$$

where the normalization constant Z is also known as the partition function, and it is defined by:

$$Z = \sum_x \prod_C \psi_C(x_C) \quad (2.4)$$

An alternative representation for a MRF is the *factor graphs*. Until now, MRFs were represented using nodes and edges, making conditional dependencies explicit, while factorization properties were kept implicit. Factor graphs introduce a second type of node, the function node, so that while the graph still has the same properties, its factorization is made explicit within the graph topology. A factor graph equivalent to Fig. 2.3b is shown in Fig. 2.3c. Factor graphs belong to what are called bipartite graphs, as they consist of two distinct kinds of nodes, and all links go between nodes of opposite type.

In this work we use the factor graph representation because when expressing the factorization properties explicitly, the correspondence between the intuition and the graph topology is more direct. More formally, a MRF is defined as a graphical model that can be represented as an undirected bipartite factor graph $G = (X, F, E)$, where each node $X_n \in X, n \in \{1, 2, \dots, N\}$ represents an S discrete-valued random variable and x_n represents the possible realizations of that random variable [10, 91, 43]. Each factor node $f_m \in F, m \in \{1, 2, \dots, M\}$ is a function mapping from a subset of variables $\{X_a, X_b, \dots\} \subseteq X, \{a, b, \dots\} \subseteq \{1, 2, \dots, N\}$ to the factor node f_m , where the relation between them is represented by edges $\{e_{\langle m, a \rangle}, e_{\langle m, b \rangle}, \dots\} \in E$ connecting each variable node $\{X_a, X_b, \dots\}$.

The joint probability is then factorized as:

$$P(X_1 = x_1, X_2 = x_2, \dots, X_N = x_N) \equiv P(x) \quad (2.5)$$

$$P(x) = \frac{1}{Z} \prod_{m=1}^M f_m(x_m), \quad (2.6)$$

where the factor f_m has an argument x_m that represents the relevant subset of variables from X . The partition function is now defined as:

$$Z = \sum_x \prod_{m=1}^M f_m(x_m). \quad (2.7)$$

It is important to note that it is assumed that functions f_m are non-negative and finite so $P(x)$ is a well defined probability distribution.

While a graph allows the relations between different variables to be easily encoded, the main aim of such modelling is to allow inference of such variables conditioned to an observation. This means to infer the most likely state for each of the unobserved nodes within the graph. To this end, it is necessary to compute the marginal probability as:

$$P_n(x_n) = \sum_{x \setminus x_n} P(x) \quad (2.8)$$

where $x \setminus x_n$ means all the realizations in the graph except the realizations for the node X_n . $P_n(x_n)$ means the probability of the states of the random variable X_n and will denote the marginal probability function obtained by marginalizing $P(x)$ onto the random variable X_n .

However the complexity of this problem grows exponentially with the number of variables N and thus becomes computationally intractable. Approximation techniques have been proposed in recent years. Examples of such algorithms are Graph Cuts and Belief Propagation (BP), among others, which are often the most feasible in practice. In this work BP has been chosen to perform inference. Section 2.3 contains a detailed description of the BP algorithm.

2.3 Belief Propagation

Belief Propagation (BP) [53, 91, 86, 27] is an iterative algorithm for performing inference on MRFs. Different approaches can be applied depending on the particularities of the problem. For instance, the *Max-Product* BP algorithm is focused on finding the maximum posterior probability for the whole graph and *Sum-Product* BP algorithm is focused on finding the most probable state for each node.

In this work, we only study the *Sum-Product* BP algorithm, because our goal is to obtain the most probable state for each pixel in the image. Therefore, it perfectly suits our needs. We will explain in the following sections how we model the image pixels in more detail.

BP algorithms are based on passing messages around the graph. The *sum-product* version uses two types of messages. The first type is those messages sent by random variable nodes and received by factor nodes. These messages are defined as:

$$q_{n \rightarrow m}(x_n) = \prod_{m' \in M(n) \setminus m} r_{m' \rightarrow n}(x_n), \quad (2.9)$$

where $M(n)$ is the set of factors in which variable X_n participates.

The second type of messages are those sent by factor nodes and received by random variable nodes. These messages are defined as:

$$r_{m \rightarrow n}(x_n) = \sum_{x_m \setminus n} \left(f_m(x_m) \prod_{n' \in N(m) \setminus n} q_{n' \rightarrow m}(x_{n'}) \right), \quad (2.10)$$

where $N(m)$ is the set of variables connected to f_m , the factor node.

In Fig. 2.6 an example of these two messages is shown. The blue arrows represent the messages noted as q in Eq. 2.9, while the red arrows represent the r messages of Eq. 2.10.

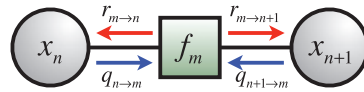


Figure 2.6: Messages involved in the belief propagation *Sum-Product* algorithm in a 1D graph with two single nodes. The blue arrows are the messages from nodes to factors and the red arrows the messages from factor to nodes.

To find the marginal probability of a node, all the incoming messages in that node are multiplied. Then we obtain a *belief* $b_n(x_n)$, which represents an approximation of the real marginal probability $P_n(x_n)$. The *belief* is computed as:

$$b_n(x_n) = \frac{1}{Z} \prod_{m \in M(n)} r_{m \rightarrow n}(x_n). \quad (2.11)$$

The computation of the marginal probability as described before is an iterative process. If the graph has a tree topology, the *belief* is guaranteed to converge to the marginal probability. In this case the process starts at the leaves and the messages are propagated according to equations 2.11 and 2.10.

If the graph contains loops, there is no equivalent guarantee of convergence, as the node is dependent on itself. For instance, any change in the node state affects the state of its neighbours which at the same time affects again the node itself. However, there is significant large empirical evidence that, even in the presence of loops, BP provides a good approximation of the optimum state. It is therefore a common practice to use BP even in the presence of loops.

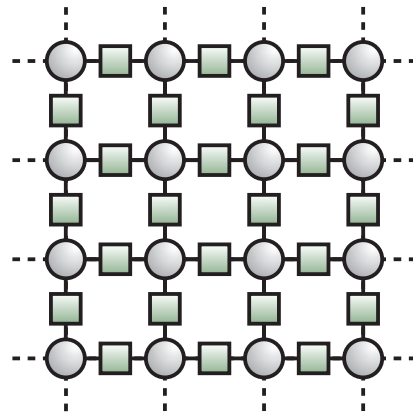


Figure 2.7: MRF with a lattice topology, where each node is connected to the nodes placed in his left, right, up and down. This is a common topology for graphs in computer vision, as it can represent the pixels in an image and reproduce their spatial layout.

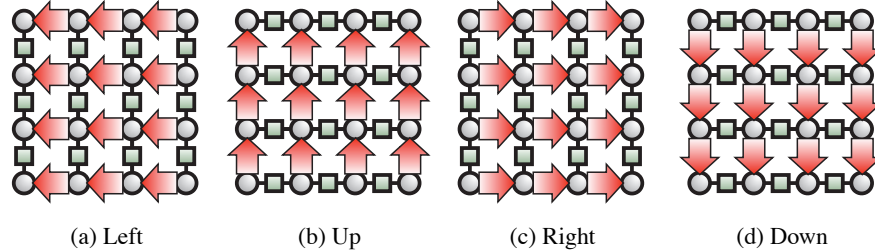


Figure 2.8: Different directions of the 4 connected Markov Random Field.

In this work we use MRF in a lattice form (see Fig. 2.7), where a node has four connections. In this case, BP can be computed by splitting the graph into the different edge directions (up, down, left, right) and then messages are propagated separately. Fig. 2.8 shows an example of a MRF split into its different edge directions. Finally all the split graphs are merged by point-wise multiplication. This process is repeated until convergence. Note that the computation of BP in this way enables a straightforward parallelization that can drastically reduce computation time.

After the network converges, the more feasible state for each node is selected. Different criteria can be used for this purpose, depending on whether we are interested in the *Maximum a Posteriori* (MAP) or the *Minimum Mean Squared Error* (MMSE).

- In the MAP case, the state x_n with higher belief $b_n(x_n)$ is directly selected as the inferred state of the node [67],

$$x_n^{MAP} = \operatorname{argmax}_{x_n} (b_n(x_n)) \quad (2.12)$$

- The MMSE is computed as a weighted mean of the different states, where the weight is defined by the belief for each state. As the states of the node are typically discrete, the closest state to the mean value is selected, so having the least squared error to the weighted mean computed[93].

$$x_n^{MMSE} = \sum_{x_n} x_n b_n(x_n) \quad (2.13)$$

2.4 Our model

Our MRF is based on that used in [93, 84, 11], where each pixel in the image (i, j) is represented by a random variable $X_{(i,j)}$. The relations between neighbouring nodes is set through factor nodes f_m forming a lattice shape (see Fig. 2.7 for a visual depiction of this topology). Factor nodes f_m are also known as a *compatibility function*. $X_{(i,j)}$ nodes, are also related

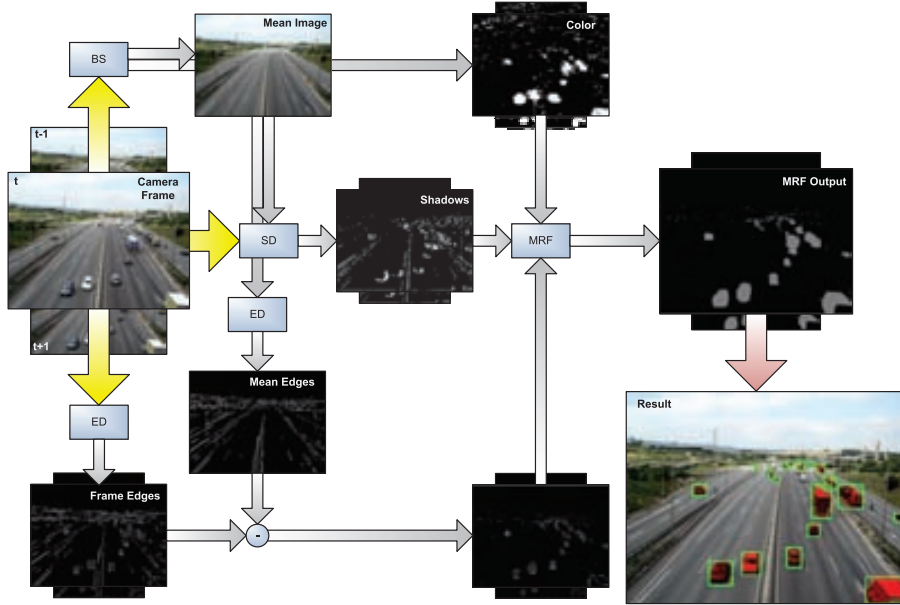


Figure 2.9: Bloc diagram of the proposed method. t indicates the frame index, BS is refers to Background Subtraction. SD refers to Shadow Detector and ED refers to Edge Detector

to another type of random variables $D_{(i,j)}$ called *observation data*. The *observation data* is extracted directly from the image and could be continuous or discrete depending on the formulation. We define the *local evidence* $h_{(i,j)}$ as the factor node that converts $D_{(i,j)}$ into a probability distribution over the states of $X_{(i,j)}$. For instance, an image where $D_{(i,j)}$ is the pixel value in a gray scale image and where $X_{(i,j)}$ have two possible states representing white and black, $h_{(i,j)}$ gives a probability to the state being white that is linearly correlated to the value of $D_{(i,j)}$.

As we stated in the introduction, our model has three information sources, all of them binarised to construct the input to the MRF. A pixel-level background subtraction method based on the RGB value of the pixel is constructed based on modelling the historical values for that pixel through a 3-dimensional Gaussian distribution. A shadow detector is used to provide binarised information of whether a pixel is obscured by a shadow. The method employed is called the Sakbot system [21], and uses the background image given by the previous source as a basis to compare the variations in the luminance and chrominance in the HSV color space. The last cue used results from applying the Sobel edge detector to the current image, and then removing the edges detected in the background image using the same edge detection method. Fig. 2.9 presents a block diagram of the whole system.

The most immediate graph structure for combining these three sources of information is shown in Fig. 2.10a. In there, each source is connected independently through functions h_a^1 , h_a^2 and h_a^3 . The main drawbacks of this design is that the formulation becomes complex and

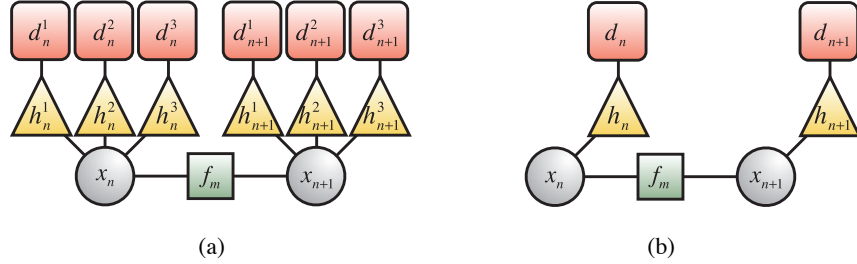


Figure 2.10: Graphical representations of the influence of the three different information sources on the graph. (a) Shows the direct representation, where each source is connected independently through three different functions. (b) The output of each binary source of information is codified to generate a single function of each state. This is the representation used in this section.

it is hardly scalable. To overcome these drawbacks, we propose to codify the output of our three binary detectors into a code of eight states, corresponding to all possible permutations. The resultant graphical model is shown in Fig. 2.10b, while table 2.1 shows the meaning of the eight states.

Shadow D.	Edge D.	Color D.	Value
			0
		x	1
	x		2
	x	x	3
x			4
x		x	5
x	x		6
x	x	x	7

Table 2.1: Codification of the different permutations of the detectors. D . means Detector.

Formally we define our model as a 4-partite graph $G = (X, D, F, H, E)$, corresponding to two types of random variable nodes (X, D), two types of factors nodes (F, H) and the edges between them (E). Each pixel (i, j) in a $w \times h$ frame is associated with two random variables, $X_{(i,j)}$ and $D_{(i,j)}$, which are related through a factor node $h_{(i,j)}$. $X_{(i,j)}$ represents a binary discrete-valued random variable with two possible states, indicating whether the corresponding pixel is assigned to the foreground or to the background. We define as $x_{(i,j)}$ the possible realizations of $X_{(i,j)}$. In the same way we define $d_{(i,j)}$ as the possible realizations of $D_{(i,j)}$, but in this case $D_{(i,j)}$ has eight possible states.

The information regarding the spatial relations is obtained by relating each node $X_{(i,j)}$ to its neighbouring nodes $X_{(l,k)} \forall (l,k) \in \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$ through factor nodes $f_{\langle (i,j), (l,k) \rangle} \in F$. In addition, our model has three layers that corresponds to three

consecutive frames from $t - 1$ to $t + 1$. This addition introduces temporal information and results in a model that also imposes temporal consistency onto the predicted background image. To distinguish the nodes in different temporal layers, a superindex indicating the frame is introduced, so the variables are now notated as $X_{(i,j)}^t$ and $D_{(i,j)}^t$. Each variable node is connected by two relations with nodes at different time layers, i.e. node variables $X_{(i,j)}^p$ where, $p \in \{t - 1, t + 1\}$. This final structure is graphically represented in Fig. 2.11.

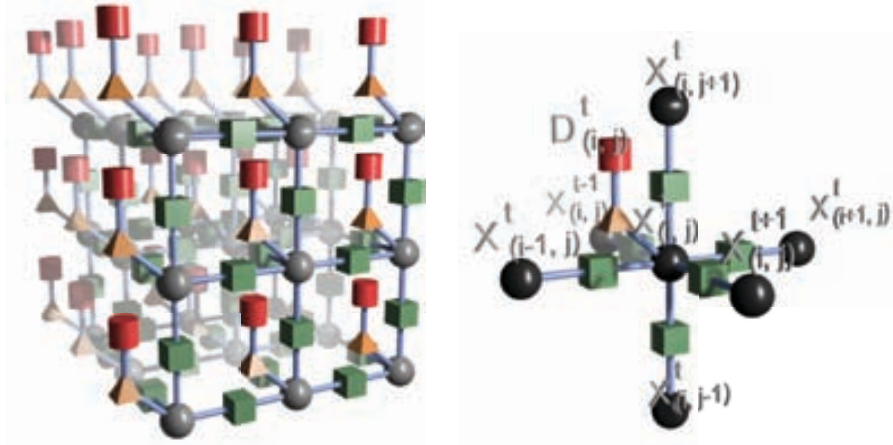


Figure 2.11: The left image corresponds to a representation of our model. Spheres represent the variable nodes belonging to X , cylinders represent the variable nodes corresponding to D , cubes represent factors from F and pyramids represent factors from H . The right image shows all the connections of one node.

In order to simplify the notation, n is used to index all the pixels in the image and substitutes the indexing by row and column. N is then defined as the total number of pixels in the image, which corresponds to the number of local evidence functions. For example, $h(x_n, d_n)$ is one of these functions, and x_n represents all possible realizations of the corresponding variable node $X_n \equiv X_{(i,j)}^t \in X$ and d_n all possible realizations of the corresponding variable node $D_n \equiv D_{(i,j)}^t \in D$. The factor nodes are represented by $f(x_o, x_u)$, where x_o and x_u represent the possible realizations of the neighbour variable nodes $\{X_o, X_u\} \equiv \{X_{(i,j)}^t, X_{(l,k)}^p\} \in X$. Finally, M is the total number of compatibility functions and $f_m(x_m)$ is one of these functions. Similarly, x_m represent the possible realizations of two random variable nodes $\{X_{(i,j)}^t, X_{(l,k)}^p\} \in X$ and is equivalent to $f(x_o, x_u)$.

Now the joint probability distribution is defined as:

$$P(X_1 = x_1, \dots, X_N = x_N, D_1 = d_1, \dots, D_N = d_N) \equiv P(x, d) \quad (2.14)$$

$$P(x, d) = \frac{1}{Z} \prod_{n=1}^N h(x_n, d_n) \prod_{m=1}^M f_m(x_m). \quad (2.15)$$

Note that the join probability function is like the MRF joint probability function, although adapted to add the observation data.

The state for each variable node D_n is fixed by the observation data. Therefore, we have to infer the optimal state for each variable node X_n . The sum-product BP equations are now defined as:

$$q_{n \rightarrow m}(x_n) = h(x_n, d_n) \prod_{m' \in M(n) \setminus m} r_{m' \rightarrow n}(x_n) \quad (2.16)$$

for messages from X_n to factor nodes f_m and,

$$r_{m \rightarrow n}(x_n) = \sum_{x_m \setminus n} \left(f_m(x_m) \prod_{n' \in N(m) \setminus n} q_{n' \rightarrow m}(x_{n'}) \right) \quad (2.17)$$

for messages from factor nodes f_m to variable nodes X_n . Finally the *belief* $b_n(x_n)$ equation becomes:

$$b_n(x_n) = \frac{1}{Z} h(x_n, d_n) \prod_{m \in M(n)} r_{m \rightarrow n}(x_n) \quad (2.18)$$

where the normalization constant Z can be dropped.

We define the local evidence $h(x_n, d_n)$ as:

$$h(x_n, d_n) = \begin{cases} [\vartheta_0, 1 - \vartheta_0]^T & \text{if } D_n = 0 \\ [\vartheta_1, 1 - \vartheta_1]^T & \text{if } D_n = 1 \\ [\vartheta_2, 1 - \vartheta_2]^T & \text{if } D_n = 2 \\ [\vartheta_3, 1 - \vartheta_3]^T & \text{if } D_n = 3 \\ [\vartheta_4, 1 - \vartheta_4]^T & \text{if } D_n = 4 \\ [\vartheta_5, 1 - \vartheta_5]^T & \text{if } D_n = 5 \\ [\vartheta_6, 1 - \vartheta_6]^T & \text{if } D_n = 6 \\ [\vartheta_7, 1 - \vartheta_7]^T & \text{if } D_n = 7 \end{cases} \quad (2.19)$$

and the compatibility matrix $f(x_o, x_u)$ as:

$$f(x_o, x_u) = \begin{cases} \theta & \text{if } X_o = X_u \\ 1 - \theta & \text{otherwise} \end{cases} \quad (2.20)$$

To obtain all the parameters in our algorithm, $\vartheta_{0, \dots, 7}$ and θ , we have made a probabilistic study using n representative frames ($I^{0, \dots, n-1}$) manually annotated. A binary mask L^k was obtained for each of them, where non-null values represents foreground.

$$\begin{aligned}\vartheta_a &= P(X_j = 1|D_j = a) \\ 1 - \vartheta_a &= P(X_j = 0|D_j = a)\end{aligned}\quad (2.21)$$

We say that ϑ_a is the prior probability of a pixel annotated as a , to being part of the foreground. Therefore, we can use Bayes theorem to compute:

$$P(X_j = 1|D_j = a) = \frac{P(D_j = a|X_j = 1)P(X_j = 1)}{P(D_j = a)} \quad (2.22)$$

$$P(X_j = 0|D_j = a) = \frac{P(D_j = a|X_j = 0)P(X_j = 0)}{P(D_j = a)} \quad (2.23)$$

Let m be the number of pixels in an image and D_i^k the value of the observation data in the pixel i on the frame k . We compute the marginals and the likelihood using the annotated frames L_k .

$$P(X_j = 1) = \frac{1}{n \cdot m} \sum_{k=0}^{n-1} \sum_{i=0}^{m-1} L_i^k \quad (2.24)$$

$$P(X_j = 0) = 1 - P(X_j = 1) \quad (2.25)$$

$$P(D_j = a) = \frac{1}{n \cdot m} \sum_{k=0}^{n-1} \sum_{i=0}^{m-1} \int_{-\infty}^{\infty} \delta(a - D_i^k) d\delta \quad (2.26)$$

$$P(D_j = a|X_j = 1) = \frac{\sum_{k=0}^{n-1} \sum_{i=0}^{m-1} L_i^k \int_{-\infty}^{\infty} \delta(a - D_i^k) d\delta}{\sum_{k=0}^{n-1} \sum_{i=0}^{m-1} L_i^k} \quad (2.27)$$

$$P(D_j = a|X_j = 0) = 1 - P(D_j = a|X_j = 1) \quad (2.28)$$

where δ is the Kronecker delta function. Finally we can compute θ as:

$$\theta = \frac{1}{n \cdot m} \sum_{k=0}^{n-1} \sum_{i=1}^m \frac{\sum_{l \in N(L_i^k)} \int_{-\infty}^{\infty} \delta(l - L_i^k) d\delta}{N_i^k} \quad (2.29)$$

where N_i^k is the number of neighbours and $N(L_i^k)$ is the set of neighbours of L_i^k .

2.5 Experimental results

As the computational cost of the inference process depends linearly on the number of iterations of the BP, we have made an empirical study to determine the cost of setting an early convergence based on the maximum number of iterations. To this end, several video sequences



Figure 2.12: The left image shows the result of performing one iteration of BP. The center image shows the result of performing five iterations. The right image shows the difference between motion masks.



Figure 2.13: Motion segmentation using an IR sequence.

were analysed, for which the BP algorithm was allowed to run for as long as it took to converge. Based on this analysis, all of the results presented hereafter use a maximum number of five iterations, which is usually enough as to arrive to a convergence point. Fig. 2.12 shows how the motion segmentation mask varies depending on the number of iteration inside the BP algorithm (in this case from 1 to 5).

Even when only five iterations are used, our implementation works at 0.8 seconds per frame. Therefore, it does not run in real time. However, our implementation is a combination of Matlab and C++. We estimate that transcoding the Matlab parts into C++ should be enough to achieve a reduction of a factor of ten in terms of computation time. In addition, BP is highly parallelizable allowing for the use of other frameworks like CUDA or OpenCL, although these implementation tasks are outside the scope of this work.

We have tested our approach using video sequences of color and IR cameras. The IR sequences were recorded using thermal infrared cameras which captures electromagnetic bands of $3 - 5\mu m$ and $8 - 12\mu m$. In this case, the shadow detector is disabled, as there are no shadows on thermal images. Some results using the IR camera are shown in Fig. 2.13.

Another test is presented in Fig. 2.14, which shows the output of each motion detector (color, edges and shadows) and the result when combining all sources using the proposed method. As it can be seen the resultant mask is much better than the others.

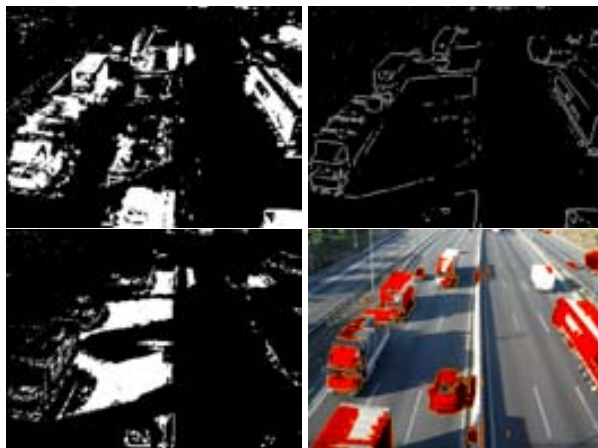
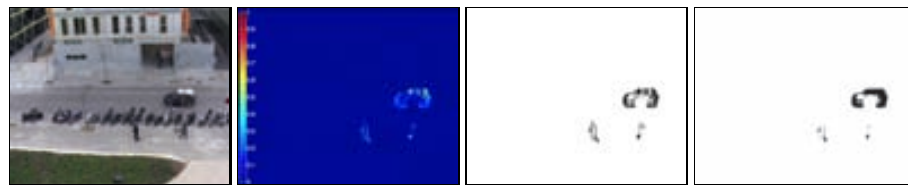
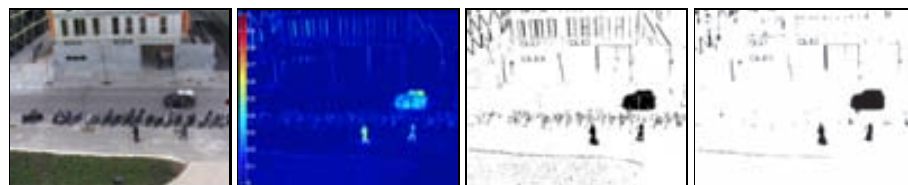


Figure 2.14: The top images are the results of the color background subtraction (left) and the edge detector (right). The bottom left shows the shadow detector result and on the right is the result after combining all previous outputs using the proposed method.

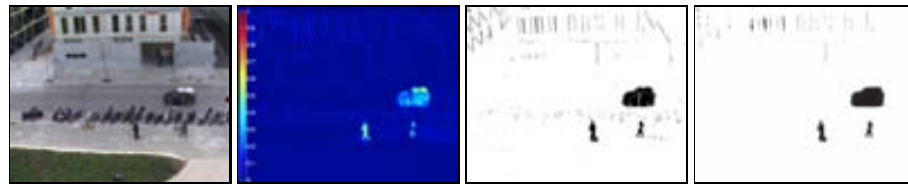
Finally, the last test presented in this chapter demonstrates that our method can be used to improve the motion mask provided by the main pixel-level background subtraction algorithms present in the literature. We have replaced the color source present in our model by the output of those methods and, as you can see in Fig. 2.15 and Fig. 2.16, the noise is reduced and the segmentation is more accurate. It should be stated, that the shadow detector was disabled again because the major part of the algorithms represents the background image in gray scale values, which is incompatible with the HSV color space required by our shadow detector.



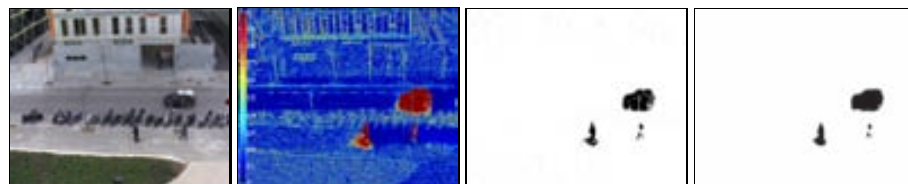
(a) Inter-Frame difference, using the previous frame.



(b) Median of the previous 11 frames for each pixel.

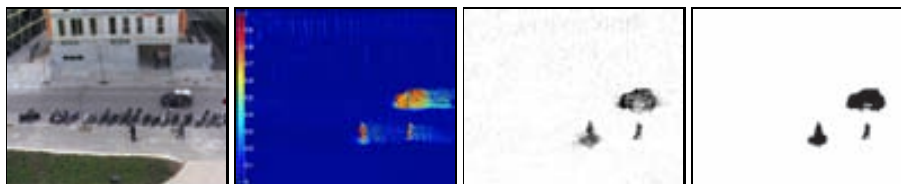


(c) Iterative Mean of the previous frames for each pixel.

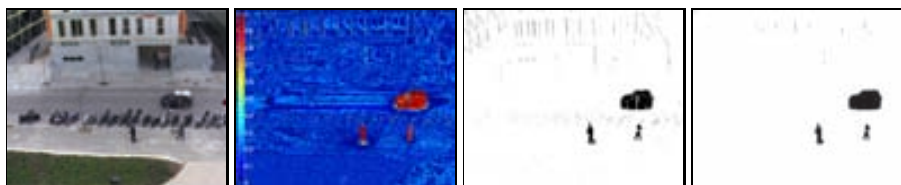


(d) A Gaussian distribution for each pixel.

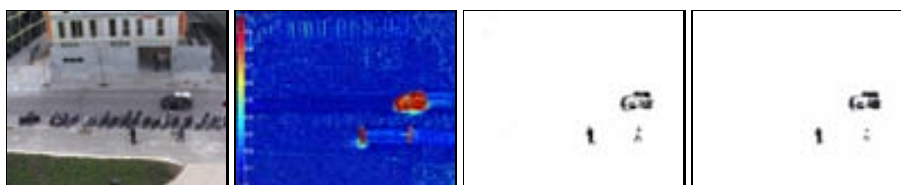
Figure 2.15: Comparison of different pixel-level background subtraction techniques, applied to the same video sequence. The first image is the original frame, the second is the difference or probability given by the algorithm, the third image is the output of the algorithm (motion mask) and the last image is the output improved by our method.



(a) A Mixture of 3 Gaussians for each pixel.



(b) Kernel Density Estimator.



(c) Mean Shift of 3 Gaussians for each pixel.

Figure 2.16: Comparison of Multi-modal background subtraction techniques at pixel-level, applied to the same video sequence. The first image is the original frame, the second is the difference or probability given by the algorithm, the third image is the output of the algorithm (motion mask) and the last image is the output improved by our method.

Chapter 3

Background Subtraction for Non-static cameras based on MKT

This chapter is dedicated to the problem of detecting motion in scenes recorded with non-static cameras. In such cases, it is not possible to use a background subtraction algorithm directly, as the motion produced by the camera can not be distinguished from the motion present in the scene. We propose a method, based on Multiple Kernel Tracking (MKT), to estimate the global motion of the video sequence. There are however some assumptions. The first is that the majority of the image area belongs to the background. Secondly, the absence of parallax is assumed. This is a typical scenario for PTZ cameras. In addition we show how to incrementally generate the background image, in order to cover the whole field of view of the camera. The proposed method is very undemanding in terms of computation time and in memory requirements, and it can run comfortably in real time without resorting to complex implementations.

3.1 Introduction

In the previous chapter we have seen how motion can be detected in a video sequence using background subtraction. The background image is intended to be a representation of the elements of a recorded scene that do not contain objects in motion. Then, regions with a large difference (under some criteria) between the frame and the background image indicate where the objects in motion are located. In order to generate the background image, all background subtraction techniques assume that the camera is completely static and therefore, the pixels forming the background image are related directly to specific regions in the scene.

However, in many practical cases it seldom happens that the camera is static. Even in the

cases where cameras are supposed to be static, cameras can vibrate due to strong wind or due to traffic of heavy vehicles for example. In such cases, pixel locations do not correspond to the scene location, making it impossible to directly generate a correct background image. In order to restore the relation between the pixel locations, it is necessary to first compensate the camera movement or, in other words, it is necessary to align all the frames in the video sequence in a common coordinate system.

When the camera motion is not very large, the alignment can be performed by selecting a reference frame and aligning the rest with respect to it. Otherwise, alignment becomes more complex and it is necessary to generate a representation of the whole scene. For instance, video sequences recorded using robotic cameras like Pan-Tilt-Zoom cameras (PTZ) are capable of generating larger 2D movement than the width of a single frame. Therefore, using only a single reference frame to align all the video sequence becomes unfeasible, as some frames do not have an image region in common. It is then necessary to stitch the newly aligned regions to the reference frame. This process is called mosaicing and will be the basis of the rest of this thesis. Note that when using the mosaic image as a background (referred to as the background mosaic), and provided that the new frame is properly aligned with the background image, then static background subtraction algorithms are directly applicable.

Multiple approaches exist in the literature targeting the use of mosaicing for detecting motion. For example, [9] uses the alignment method proposed in [80] to find a set of point correspondences between the current frame and a reference frame. Then an affine transformation is estimated with the least square error between the set of point pairs. In this case the background mosaic is generated by simply computing the mean of each pixel. Another work presented in [83] proposes to apply an affine transformation, computed using an Iterated Re-weighted Least Squares method on a multi-resolution image pyramid. The authors of [2, 7] propose a method also based on point correspondences. However, in these cases the point correspondence is improved (in terms of computation time) by using phase correlation. Again, the mean is used to model the background mosaic, but in this case the frames are aligned using a projective transformation. In [95] the frame alignment is speeded up using the GPU to compute point correspondences and then the optimal alignment is computed through RANSAC (RANdom SAmple Consensus [28]), making the result robust to outliers and points that do not have a real correspondence. The optimal alignment can also be performed by using the the MLESAC (m-estimator sample consensus) algorithm [81], which is a robust estimator that improves the RANSAC algorithm, and is used in [36] to align the frames. Finally, in [31] a probability density function, indicating the foreground regions, is computed by examining the key point matching and failures between each frame and the background mosaic (in this case a set of registered images, where each image corresponds to a predefined position of the PTZ camera). Note that only the key points and its descriptors are need to be keep. They use the SURF algorithm (Speeded Up Robust Features [6]), to obtain a dense set of key points.

This chapter aims to extend the applicability of the work presented in the previous chapter to cases where the cameras are not static. The background subtraction algorithm presented in

the previous chapter is computationally expensive. Therefore, the computational complexity of the alignment method has to be low enough to be executed well within real-time. To this end, we use the Multiple Kernel Tracking framework, which is a well known, efficient and accurate tracking algorithm. However, in our case, instead of tracking a region of interest, for example a moving object, we propose to track the whole frame. The underlying assumptions are that most of the frame will belong to the background and the scene does not contain parallax effects.

The remainder of the chapter is as follows. First, a detailed introduction to Kernel Tracking methods is presented in section 3.2. Section 3.3 describes the procedure to construct and update the background mosaic over time. Finally, section 3.4 presents the experimental results for different video sequences.

3.2 Kernel Tracking

Kernel Tracking algorithms are based on the idea of representing the target region with a color/intensity histogram, where the contribution of each pixel is spatially weighted by a kernel function. Through the use of a kernel function, the representation incorporates spatial information, which is lost when generating an unweighted histogram. A histogram representation is extremely efficient, as only a few bins are needed to represent a target; it is tolerant to some degree of aspect variations, such as rotations, missalignments or scale variations. The computation of a histogram boils down to the (weighted) addition of the pixel values within an image area, therefore being very quick to compute.

In this section we will introduce in detail the two main Kernel Tracking algorithms. The Mean Shift Tracking, which was the first Kernel based tracking algorithm and the Multiple Kernel Tracking, which is the method used in this work.

3.2.1 Mean Shift Tracking

Comaniciu et al. introduced the first Kernel Tracking method in [19], when they adapted the Mean Shift algorithm, previously used only for clustering, to the problem of tracking. The localization estimation problem is solved by iteratively optimizing the similarity between a target region \mathbf{q} and a candidate region \mathbf{p} using a gradient descent technique. The target region is represented by a kernel-weighted histogram, $\mathbf{q} = (q_1, q_2, \dots, q_m)^t$ and is computed as:

$$q_u = \mathbf{N} \sum_{i=1}^n K(\mathbf{x}_i - \mathbf{c}) \delta(b(\mathbf{x}_i, t), u), \quad (3.1)$$

where $K(\mathbf{x}_i - \mathbf{c}) = k(\|\mathbf{x}_i - \mathbf{c}\|^2)$ is a non-negative, non-increasing and piecewise differentiable

isotropic kernel function centred at \mathbf{c} , $b(\mathbf{x}_i, t)$ represents the bin assigned to the pixel intensity value at location \mathbf{x} , δ is the Kronecker delta function and \mathbf{N} is a normalization constant that implies $\sum_{u=1}^m q_u = 1$ and is defined as:

$$\mathbf{N} = \frac{1}{\sum_{i=1}^n K(\mathbf{x}_i - \mathbf{c})}. \quad (3.2)$$

Note that if the kernel function is normalized, i.e. $\sum_{i=1}^n K(i) = 1$, then the resulting histogram is normalised as well and \mathbf{N} can be discarded.

Following the notation later introduced by Hager et al. [33], we can rewrite equation 3.1 in a more compact form as:

$$\mathbf{q} = \mathbf{U}^t \mathbf{K}(\mathbf{c}), \quad (3.3)$$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m]$ is a n by m *sifting matrix* and \mathbf{u} is a *sifting vector* defined as $\mathbf{u}_i = \mathbf{u}_i(t) = \delta(b(\mathbf{x}_i, t), u)$. Similarly, the kernel function is re-defined as $\mathbf{K} = \mathbf{K}_i(c) = K(\mathbf{x}_i - \mathbf{c})$. In the same way, the candidate region \mathbf{p} at time t' and centred about \mathbf{c} is given by:

$$\mathbf{p}(\mathbf{c}) = \mathbf{U}^t \mathbf{K}(\mathbf{c}). \quad (3.4)$$

The minimisation problem can be formulated as finding the location \mathbf{c}^* that minimizes the error function between the two histograms. The Bhattacharyya coefficient is used in [19], and it is defined as $\sqrt{\mathbf{p}(\mathbf{c})} \sqrt{\mathbf{q}}$, where the square root operator is applied component-wise to the vector argument. Then using Taylor series to expand the Bhattacharyya coefficient about \mathbf{p} , we obtain the expression to maximize

$$O(\mathbf{c}) = \mathbf{w}^t \mathbf{K}(\mathbf{c}), \quad (3.5)$$

where \mathbf{w} is defined as:

$$\mathbf{w} = \mathbf{U} \left(\frac{\sqrt{\mathbf{q}}}{\sqrt{\mathbf{p}(\mathbf{c})}} \right), \quad (3.6)$$

where the division is again component-wise to the associated vectors. Then the similarity is optimized by computing the gradient and setting it to zero. The final solution, written in the form of weighted mean, is:

$$\Delta \mathbf{c} = \frac{\sum_{i=1}^n (\mathbf{x}_i - \mathbf{c}) w_i g(\|\mathbf{x}_i - \mathbf{c}\|^2)}{\sum_{i=1}^n w_i g(\|\mathbf{x}_i - \mathbf{c}\|^2)} \quad (3.7)$$

where $\Delta \mathbf{c} = \mathbf{c}^* - \mathbf{c}$ and $g(x) = -k'(x)$.

For further details of this method, including the demonstration of why the Mean Shift vector approximates the gradient of the function, can be found in [19].

3.2.2 Multiple Kernel Tracking with SSD

Multiple Kernel Tracking with SSD (MKT) [33], introduces some improvements with respect to the classical Mean Shift algorithm. In this approach the Bhattacharyya coefficient is replaced by the sum of squared differences (SSD) error function:

$$O(\mathbf{c}) = \|\sqrt{\mathbf{q}} - \sqrt{\mathbf{p}(\mathbf{c})}\|^2, \quad (3.8)$$

So instead of maximizing the Bhattacharyya coefficient, our goal is to minimize the SSD error. To optimize the objective function (Eq. 3.8), Hager et al. applied Taylor series to expand the expression $\sqrt{\mathbf{p}(\mathbf{c})}$ giving the equation:

$$\sqrt{\mathbf{p}(\mathbf{c} + \Delta\mathbf{c})} = \sqrt{\mathbf{p}(\mathbf{c})} + \frac{1}{2}d(\mathbf{p}(\mathbf{c}))^{-\frac{1}{2}}\mathbf{U}^t\mathbf{J}^{\mathbf{K}}(\mathbf{c})\Delta\mathbf{c}, \quad (3.9)$$

where $d(\mathbf{p}(\mathbf{c}))$ denotes the matrix $m \times m$ (m number of bins) with $\mathbf{p}(\mathbf{c})$ in its diagonal and $\mathbf{J}^{\mathbf{K}}$ is the Jacobian matrix of the kernel function \mathbf{K} , which is defined by:

$$\mathbf{J}^{\mathbf{K}} = \begin{bmatrix} \frac{\partial \mathbf{K}}{\partial \mathbf{c}_x} & \frac{\partial \mathbf{K}}{\partial \mathbf{c}_y} \end{bmatrix} = \begin{bmatrix} \nabla_c K(\mathbf{x}_1 - \mathbf{c}) \\ \nabla_c K(\mathbf{x}_2 - \mathbf{c}) \\ \vdots \\ \nabla_c K(\mathbf{x}_n - \mathbf{c}), \end{bmatrix} \quad (3.10)$$

By using the notation $\frac{1}{2}d(\mathbf{p}(\mathbf{c}))^{-\frac{1}{2}}\mathbf{U}^t\mathbf{J}^{\mathbf{K}}(\mathbf{c})$ as \mathbf{M} , equation 3.9 can be written as:

$$\sqrt{\mathbf{p}(\mathbf{c} + \Delta\mathbf{c})} = \sqrt{\mathbf{p}(\mathbf{c})} + \mathbf{M}\Delta\mathbf{c}, \quad (3.11)$$

Now, by substituting in SSD objective function and then equalling to zero, this results in a standard least squares solution:

$$\Delta(\mathbf{c}) = \mathbf{M}^+(\sqrt{\mathbf{q}} - \sqrt{\mathbf{p}(\mathbf{c})}), \quad (3.12)$$

where \mathbf{M}^+ is the pseudo-inverse matrix $((\mathbf{M}^t\mathbf{M})^{-1}\mathbf{M}^t)$.

Furthermore, Hager et al. also introduced the use of multiple kernels in order to represent the target. The representation in this case is based on a histogram concatenation. Each kernel in our target is stacked in a single vector $\mathbf{Q} = (\mathbf{q}_1^t, \mathbf{q}_2^t, \dots, \mathbf{q}_l^t)^t$, where l is the number of regions. The representation of the candidate region is constructed in the same way, so we note $\mathbf{P}(\mathbf{C}) = (\mathbf{p}(\mathbf{c}^1)^t, \mathbf{p}(\mathbf{c}^2)^t, \dots, \mathbf{p}(\mathbf{c}^l)^t)^t$, where $\mathbf{C} = \{\mathbf{c}^1, \mathbf{c}^2, \dots, \mathbf{c}^l\}$ is the set of region centers (or kernels center). Then equation 3.11 becomes:

$$\sqrt{\mathbf{P}(\mathbf{C} + \Delta\mathbf{c})} = \sqrt{\mathbf{P}(\mathbf{C})} + \mathbf{W}\Delta\mathbf{c}, \quad (3.13)$$

where

$$\mathbf{W} = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \\ \vdots \\ \mathbf{M}_l \end{bmatrix} \quad (3.14)$$

is all target regions \mathbf{M} 's, stacked in a single matrix of $2 \times m * l$ and the SSD objective function set to zero is again a standard least squares solution:

$$\Delta(\mathbf{c}) = \mathbf{W}^+(\sqrt{\mathbf{Q}} - \sqrt{\mathbf{P}(\mathbf{C})}). \quad (3.15)$$

Now $\Delta(\mathbf{c})$ is related to the displacement over the x -axis and y -axis, which is in fact $\Delta(\mathbf{c}) = (\Delta(\mathbf{c}_x), \Delta(\mathbf{c}_y))$. However, with multiple kernels it is possible to extend it to recover richer models. In this work we extend this model to be able to estimate rotations θ and escalations λ . In such cases the target region is defined as:

$$\mathbf{q}(\mathbf{c}_x, \mathbf{c}_y, \theta, \lambda) = \mathbf{U}^t \mathbf{K}(\mathbf{c}_x, \mathbf{c}_y, \theta, \lambda). \quad (3.16)$$

Following the same steps of the Taylor series, now the kernel Jacobian includes derivations with respect to θ and λ ,

$$\mathbf{J}^{\mathbf{K}} = \left[\frac{\partial \mathbf{K}}{\partial \mathbf{c}_x}, \frac{\partial \mathbf{K}}{\partial \mathbf{c}_y}, \frac{\partial \mathbf{K}}{\partial \theta}, \frac{\partial \mathbf{K}}{\partial \lambda} \right] = \left[\mathbf{J}_x^{\mathbf{K}}, \mathbf{J}_y^{\mathbf{K}}, \mathbf{J}_\theta^{\mathbf{K}}, \mathbf{J}_\lambda^{\mathbf{K}} \right]. \quad (3.17)$$

Derivations with respect to θ and λ are performed using the chain rule. θ is defined as:

$$\frac{\partial \mathbf{K}}{\partial \lambda} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_{(x,y)}} \frac{\partial \mathbf{c}_{(x,y)}}{\partial \theta} \quad (3.18)$$

and in the same way λ is defined as:

$$\frac{\partial \mathbf{K}}{\partial \lambda} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_{(x,y)}} \frac{\partial \mathbf{c}_{(x,y)}}{\partial \lambda}. \quad (3.19)$$

In order to differentiate the kernel with respect to the angle, we first define the 2D rotation as:

$$R(x,y) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix} + \begin{pmatrix} I_x \\ I_y \end{pmatrix} \quad (3.20)$$

where (I_x, I_y) is the image center. Then, by deriving equation 3.20 with respect to θ , we obtain,

$$\frac{\partial R(x,y)}{\partial \theta} = \begin{pmatrix} -\sin(\theta) & -\cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{pmatrix} \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix}. \quad (3.21)$$

Note that the presence of trigonometric functions makes it difficult to derive with respect to θ . By taking advantage of the iterative nature of the Multi Kernel Tracking, we can simplify equation 3.21 by assuming that $\theta \approx 0$. Therefore, $\sin(\theta) \approx 0$ and $\cos(\theta) \approx 1$. Then equation 3.21 becomes:

$$\frac{\partial R(x,y)}{\partial \theta} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix} = \begin{pmatrix} -\mathbf{c}_y + I_y \\ \mathbf{c}_x - I_x \end{pmatrix}. \quad (3.22)$$

Finally by applying the chain rule we obtain the derivative of the kernel with respect to the angle,

$$\mathbf{J}_{\theta}^{\mathbf{K}} = [\mathbf{J}_x^{\mathbf{K}}, \mathbf{J}_y^{\mathbf{K}}] \begin{pmatrix} -\mathbf{c}_y + I_y \\ \mathbf{c}_x - I_x \end{pmatrix}. \quad (3.23)$$

The chain rule is applied similarly for the case of the scaling factor, so it is defined as:

$$S(x,y) = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix} + \begin{pmatrix} I_x \\ I_y \end{pmatrix} \quad (3.24)$$

and its derivative as:

$$\frac{\partial S(x,y)}{\partial \lambda} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix} = \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix} \quad (3.25)$$

Finally, by applying again the chain rule we get the final equation:

$$\mathbf{J}_{\lambda}^{\mathbf{K}} = [\mathbf{J}_x^{\mathbf{K}}, \mathbf{J}_y^{\mathbf{K}}] \begin{pmatrix} \mathbf{c}_x - I_x \\ \mathbf{c}_y - I_y \end{pmatrix}. \quad (3.26)$$

In the next section we explain how Multiple Kernel Tracking is applied in order to align the frames.

3.3 Global Tracking Model and Maintenance

We assume that the recorded scene has a negligible parallax effect; for instance, when a PTZ camera is used, or the scene is recorded from a large-enough distance. In addition, we expect a higher proportion of background than foreground for each frame. Under this conditions, we find the camera motion by tracking the whole frame using Kernel Tracking. Compensating the camera motion allows us to apply the background subtraction method presented in the



Figure 3.1: Target regions distributed in a uniform lattice over the frame. Target regions are represented by green circles. The frame margin is indicated by the dashed rectangle.

previous chapter.

Selecting a single, large target region at the center of the frame is not a good solution. The precision of the alignment and the computation time is proportional to the size of the target region when using Kernel Tracking. In addition, we do not know a priori which parts of the frame belong to the background and which ones belong to the foreground (it is possible that the foreground is present at the center of the frame). For this reason, we propose to select multiple target regions uniformly placed over the frame, so that the whole frame is covered. Fig. 3.1 shows an example of how the target regions are distributed over a frame. In this image the target regions are represented by circles and the dashed box indicates the margin between the frame borders and the target regions. This margin is used to avoid placing target regions outside the frame, which may cause errors.

The coordinates of target regions $\mathbf{q}_u \in \mathbf{Q}$ are constant during the whole process and they are relative to a reference frame f_r , usually the first frame. Additionally, the coordinates of candidate regions $\mathbf{p}(\mathbf{c}_u) \in \mathbf{P}(\mathbf{c})$ over f_i changes during the optimization. In other words, f_r is used to incrementally align the rest of frames f_i in the video sequence. The initial camera movement hypothesis, which defines where to initially place the target regions on frame f_i , is in the result of the previous optimization over f_{i-1} . This is a simple assumption, and other more sophisticated methods can be applied, by using an autoregressive model to improve the convergence speed. Fig. 3.2 shows an example of background estimation when the camera has motion caused by heavy traffic. Fig. 3.2a shows the result of compensating the camera using MKT, while in Fig. 3.2b the result when the camera is not compensated is shown. These figures show the background model, the initial frame and its difference. It is possible to see how, without compensating the image, the background is not generated correctly.

In order to deal with: target regions placed in the foreground; lighting variations in the scene, like those caused by solar movement; and background changes. We continuously update the target region histograms with the information obtained from each frame. To this end, each bin $q_u^j \in \mathbf{q}_j$ is represented by a Gaussian $\eta_u^j(\mu_u^j, \sigma_u^j)$. The Gaussian is estimated

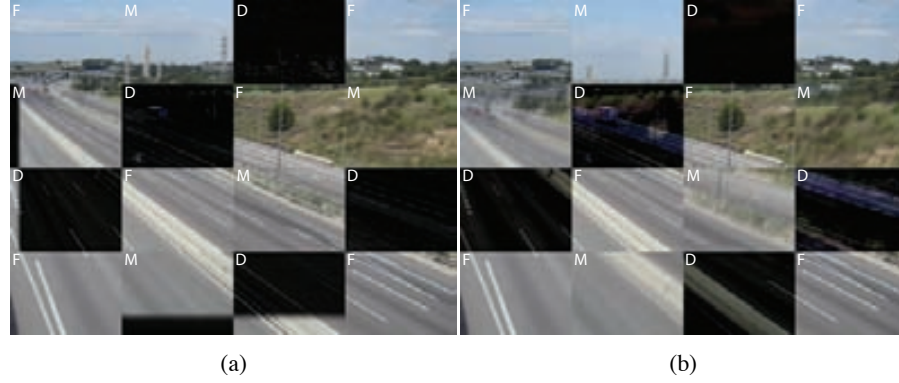


Figure 3.2: Video sequence recorded with a camera vibrating due to traffic. The right image (b) shows the result of modelling the background image directly and the left (a) image shows the result of modelling the background image using motion compensation. Each image is composed of 16 pieces, with the same size, corresponding to the first frame in the video sequence (F), the generated background model (M) and the difference between the first frame and the background model (D). It can be seen that (a) correctly generates the background model, while (b) generates a blurry background.

using an incremental procedure proposed in [45], which uses two learn factors α_1 and α_2 , where $\alpha_1 \gg \alpha_2$. α_1 is used to update the bin value if the candidate region fits with its Gaussian and α_2 is used otherwise. Using two learn factors minimizes the problem of learning from the foreground. Then each bin is defined as:

$$\mu_u^j = \mu_u^j + (\alpha_1(1 - B_t) + \alpha_2 B_t)(p(\mathbf{c}'_j)_u - \mu_u^j) \quad (3.27)$$

$$\sigma_u^{j2} = \sigma_u^{j2} + (\alpha_1(1 - B_t) + \alpha_2 B_t)((p(\mathbf{c}'_j)_u - \mu_u^j)^2 - \sigma_u^{j2}) \quad (3.28)$$

where \mathbf{c}'_j is the center of the region j after applying MKT, and B is the mask that determines if the bin fits into the model or not, which uses the Mahalanobis distance.

Until now, we have seen how to compensate the camera motion for small displacements like camera vibrations. However, if the camera has larger displacements, using a single frame to perform the alignment is not enough; for instance, a camera panning 360° . In such a case, some frames in the video sequence are not represented by any target region. That is to say, no region of f_r is present in those frames. For this reason, it is necessary to add new target regions to our model \mathbf{Q} in order to cover all new regions present in the scene. Note that now it is necessary to select a subset $\mathbf{Q}_i \subseteq \mathbf{Q}$ to perform the Kernel Tracking optimization, depending on the position of frame f_i inside the space f_r .

In order to select the target regions that fit inside f_i , for all target regions $\mathbf{q}_u \in \mathbf{Q}$, we also keep their coordinates with respect to the space f_r . Those candidate regions that fit inside



Figure 3.3: Example of target region configuration. The red box represents the area occupied by the current frame in the f_r space, the dashed box is the margin (only target regions that fit inside this box are selected), purple circles represent target regions of \mathbf{Q} that are not selected, green circles represent target regions selected to align the current frame (and they will be updated) and white circles represent target regions that added to \mathbf{Q} in this iteration.

the previous frame f_{i-1} aligned into the space f_r are selected to generate the \mathbf{Q}_i set. Fig. 3.3 shows an example of how to perform that selection. Note that now the space f_r is bigger than the size of a single frame. After performing the alignment \mathbf{Q}_i is updated using the last \mathbf{P}_i and new target regions are added to \mathbf{Q} if they cover unrepresented regions of the scene. New target regions are generated by creating a uniform lattice of possible target regions over f_i and those regions which are not present in \mathbf{Q} are added. Note that the memory requirements are very low, because we only have to keep, for each target region, the relative position to f_r and, for each bin, the mean and the variance.

In order to increase the accuracy of our system we use a mask D that sets all bins to zero that belong to target regions that have the sum of their bin variances greater than a threshold. High variance means that the color values of those regions constantly change over the time. Therefore, it is highly probable that these regions belong to the foreground. Using this mask, only the more constant target regions are used. The final equation to estimate translations is:

$$\Delta \mathbf{c} = \mathbf{W}^+(\sqrt{D\mathbf{Q}'} - \sqrt{D\mathbf{P}'(\mathbf{c})}) \quad (3.29)$$

and if we want to add rotation and escalation it becomes:

$$\Delta(\mathbf{c}, \boldsymbol{\theta}, \lambda) = \mathbf{W}^+(\sqrt{D\mathbf{Q}'} - \sqrt{D\mathbf{P}'(\mathbf{c}, \boldsymbol{\theta}, \lambda)}). \quad (3.30)$$

3.4 Experimental Results

In order to prove the robustness and efficiency of our method, we have recorded different video sequences of indoor and outdoor scenes, which include the typical movement of PTZ cameras (Pan, tilt and zoom). These videos were recorded using a camera fixed on a tripod and the camera movement was generated manually; therefore the camera movement is unpredictable. The frame resolution used is 640×480 pixels. In addition, it must be said that the videos have a high interlacing effect.

Pre-loading a video into memory, we are able to align 150 frames per second using an Intel Core 2 Duo at 2GHz. With such a high frame rate, the bottle neck is the capture time of our camera. Note that we do not use threads, OpenCL or CUDA. Therefore, we think that the frame rate can be increased significantly with a better implementation.

In Fig. 3.4 and Fig. 3.5 two examples are shown corresponding to indoor and outdoor scenes. The top image show the background mosaic generated for this sequence. The bottom images shows, from left to right: the original frame; the result obtained using a Gaussian background subtraction; the motion mask once we have applied the Markov Random Field background subtraction method proposed in the previous chapter; the motion mask overlapped with the frame. Using the same video sequences in Fig. 3.6 and Fig. 3.7 the result obtained for a set of consecutive frames are shown. Note that it is possible to appreciate that the background is moving in the same direction as the foreground.

Finally, in Fig. 3.8 and Fig. 3.9, two different states of our method (frame 923 and frame 2473) are shown in a long term video sequence. It is appreciable how our method has evolved to describe new regions. The top figures are the target regions and their locations over the mosaic image. The middle images are the Gaussian background mosaic, where the left image is the mean mosaic and the right image is the variance mosaic. Finally the bottom images show, from left to right, the original frame, the motion mask obtained with the Gaussian background subtraction and the result after using our Markov Random Field background subtraction method proposed in the previous chapter.



Figure 3.4: Example of an indoor scene. The top image shows the background mosaic generated for this sequence. The bottom images show, from left to right: the original frame; the result obtained using a Gaussian background subtraction; the motion mask once we have applied the Markov Random Field background subtraction method proposed in the previous chapter; and the motion mask overlapped with the frame.



Figure 3.5: Example of an outdoor scene. The top image shows the background mosaic generated for this sequence. The bottom images show, from left to right: the original frame; the result obtained using a Gaussian background subtraction; the motion mask once we have applied the Markov Random Field background subtraction method proposed in the previous chapter; the motion mask overlapped with the frame.

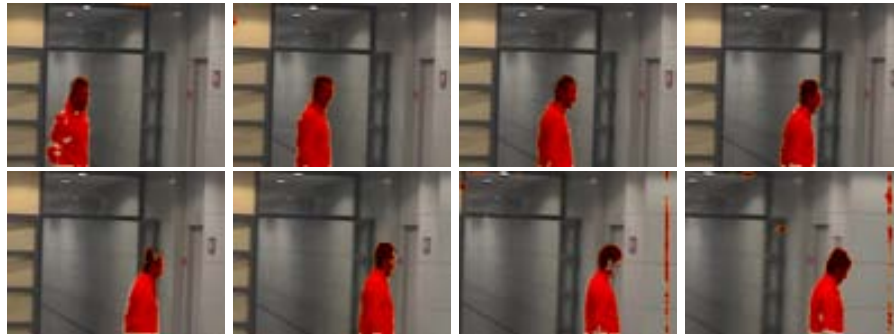


Figure 3.6: Results obtained in a set of consecutive frames (indoor video sequence). Note that the camera movement can be appreciated by comparing the foreground of the first and the last frame in the sequence.



Figure 3.7: Results obtained in a set of consecutive frames (outdoor video sequence). Note that the camera movement can be appreciated by comparing the foreground of the first and the last frame in the sequence.

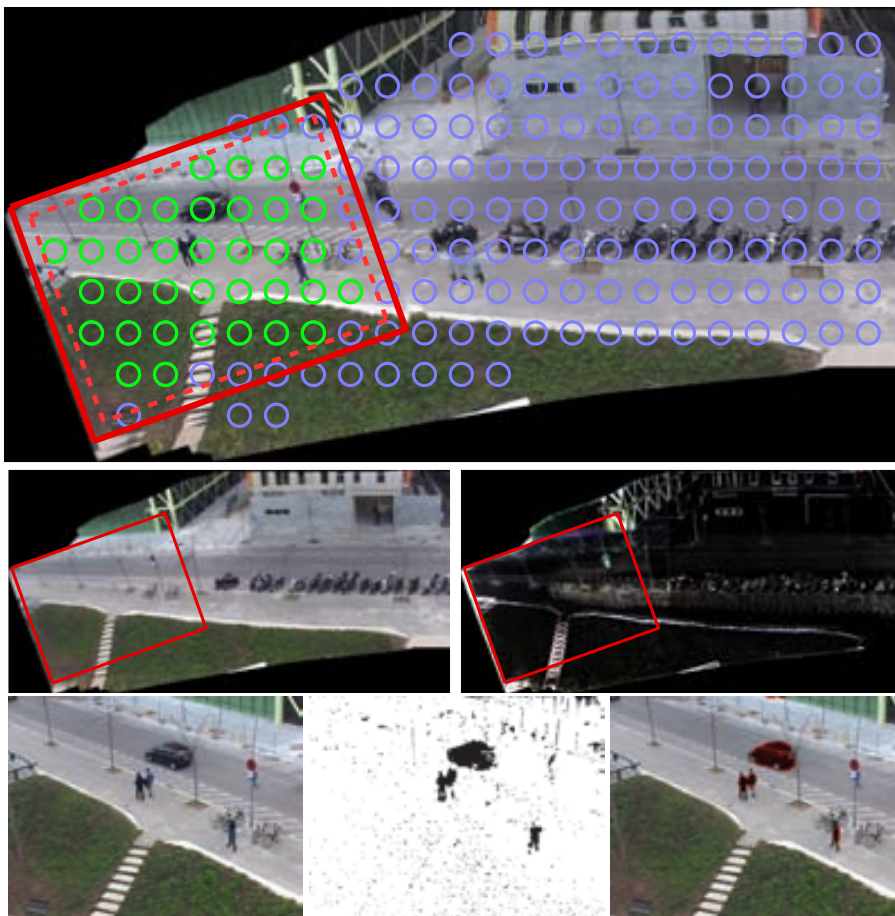


Figure 3.8: State of frame 923 in a long term video sequence. The top figure is the target regions and their locations over the mosaic image. The middle images are the Gaussian background mosaic, where the left image is the mean mosaic and the right image is the variance mosaic. Finally, the bottom images show, from left to right, the original frame, the motion mask obtained with the Gaussian background subtraction and the result after using our Markov Random Field background subtraction method proposed in the previous chapter.

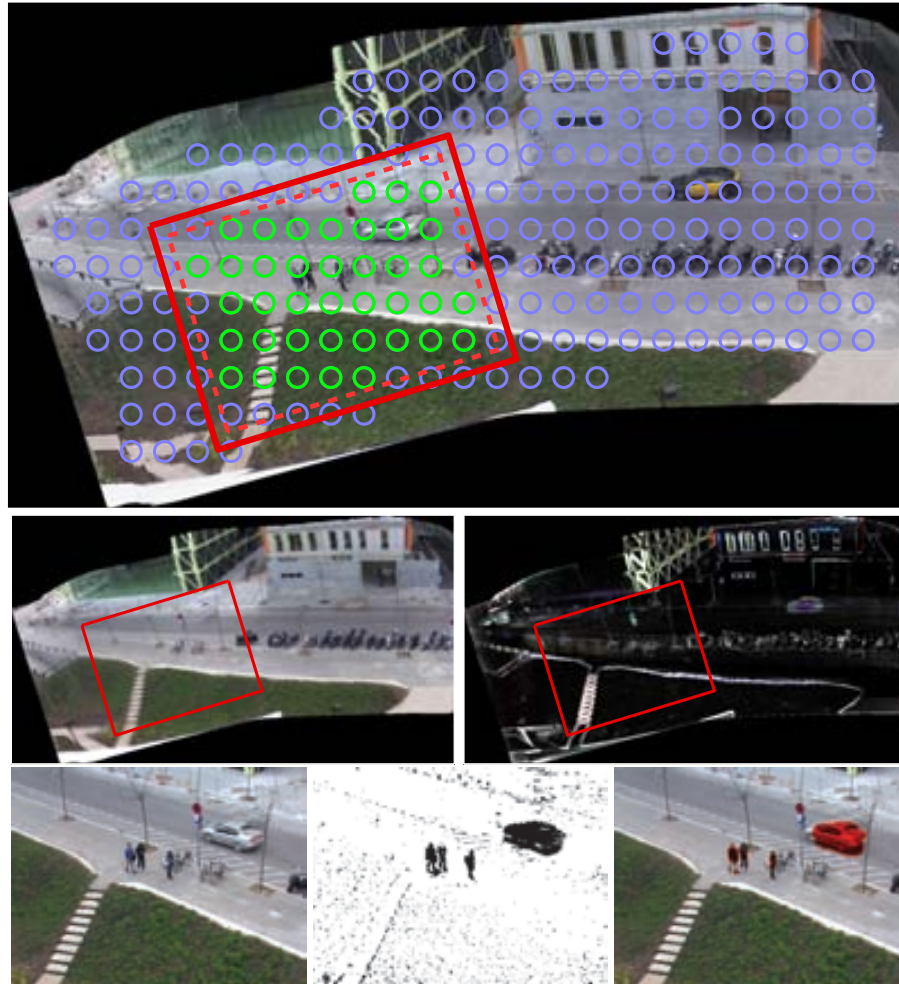


Figure 3.9: State of frame 2471 in a long term video sequence. The top figure is the target regions and their locations over the mosaic image. The middle images are the Gaussian background mosaic, where the left image is the mean mosaic and the right image is the variance mosaic. Finally, the bottom images show, from left to right, the original frame, the motion mask obtained with the Gaussian background subtraction and the result after using our Markov Random Field background subtraction method proposed in the previous chapter.

Chapter 4

DLIG: Direct Local Indirect Global alignment for Video Mosaicing

In this chapter we present a framework for real-time mosaicing for video sequences recorded from an uncalibrated PTZ camera based on multi-frame registration. To this end, a new frame alignment algorithm, the Direct Local Indirect Global (DLIG), is presented. The key idea of the DLIG alignment is to divide the frame alignment problem into the problem of registering a set of spatially related image patches. The registration is iteratively computed by sequentially imposing a good local match and global spatial coherence. The patch registration is performed using a tracking algorithm, so a very efficient local matching can be achieved. We use the patch-based registration to obtain multi-frame registration, using the mosaic coordinates to relate the current frame to patches from different frames that partially share the current field of view. Multi-frame registration prevents the error accumulation problem, one of the most important problems in mosaicing. We also show how to embed a Kernel Tracking algorithm in order to obtain a precise and extremely efficient mosaicing algorithm. Finally, we perform a quantitative evaluation of our algorithm, including a comparison with other alignment approaches, and studying its performance against interlaced videos and illumination changes.

4.1 Introduction

In the last chapter, we have seen how mosaicing can be used for detecting motion, when the camera is non static. However, as well as motion detection, mosaics are useful for many other applications.

Classically, mosaics have been used for video compression. In [40], Irani *et al.* describe techniques of video compression for video-storage and video transmission applications, which involve generating the mosaic image of the scene from the video sequence and encoding the residuals of the frames relative to the mosaic image. In [47], Lee *et al.* improve

the video coding efficiency by exploiting the layered representation in the context of MPEG-4. There, the mosaic image is a layer composed by the pixels in an object visible through the entire scene, which is called a sprite.

More recently, mosaics have been used to generate a representation of the terrain using videos recorded from the sky [16, 59] or underwater [30], for video indexing [39, 1] or for video surveillance [35, 58, 18, 97]. Two recent applications built upon a mosaicing algorithm are dynamic narratives [20] and dynamosaicing [69]. Dynamic narratives, depict the motion of one or several actors over time and is designed to produce compact, coherent and interactive narratives. In the case of dynamosaicing, the whole scene is summarized by automatically generating dynamic mosaics, ignoring the chronological order of the actions in a video sequence and focusing on the actions themselves.

There are two basic elements of a mosaicing algorithm: alignment and stitching. In this chapter we only focus on the first one. The process of frame alignment consists of estimating the transformation that places each frame in correspondence with the frames already aligned in a shared coordinate system. Depending on how the alignment is performed, it is possible to categorize the mosaicing algorithms present in the literature according two different criteria. First, they can be categorized with respect to the information used to estimate the alignment, resulting in a division between direct or featureless methods, and indirect or feature-based methods [13].

Direct methods are those which use all the available image pixels to compute an image-based error, which is then optimized to compute the alignment. The results can be accurate, but they require an already accurate initialization and typically result in high computation costs. When the input is a video sequence, accurate initialization can be assumed by using the previous alignment for initialization. In a video sequence, consecutive frames have a large overlapping region. This means that the transformations that places two consecutive frames into the mosaic are similar, which we call *frame locality*.

For instance, in [88] the author proposes to align the frames using an affine transformation obtained by an Iterated Re-weighted Least Squares method using all the pixels in the frame, on a multi-resolution image pyramid. [5] presents an alignment method based on decomposition of the frames by complex wavelet transform using the whole image. Finally, in [87] the authors propose to align the frames using the XOR Correlation on image pyramids.

In contrast, indirect methods compute the alignment by using a set of image locations. The point locations are selected to have representative local image properties so they are likely to be detected independently of the point of view of the camera. Finally, the transformation is computed as the one maximizing the matching between both the set of locations over the current frame and the set of locations over either another frame or the mosaic. Some examples of this family are [13, 85]. Both extract a set of SIFT features from all the images and then use algorithms for eliminating outliers and identifying the correspondences between images. In particular, the first one uses a probabilistic model based on RANSAC while the

second uses a geomorphic reverse measurement. In [38], instead of SIFT features they use SURF, but again they find the optimal transformation using RANSAC.

Feature-based methods are very interesting since they provide good performance and can estimate complex transformations. They are suited to mosaicing from a set of images that do not necessarily belong to a video sequence, since they do not require initialization as direct methods do. However, they lack accuracy. Furthermore, for the case of video mosaicing, direct methods make better use of the frame locality property, transforming the frame alignment problem from global search to local estimation.

The second distinction between the existing mosaicing algorithms can be made in terms of whether each frame is put in correspondence with the previous frame of the sequence or directly with the mosaic. These approaches are called frame-to-frame alignment and frame-to-mosaic alignment, respectively.

The frame-to-frame alignment approach consists of registering the current frame with the previous one, obtaining its alignment by accumulating the inter-frame registration with the previous alignment. This is the most common approach in the mosaicing literature, but it has an important drawback. Since each inter-frame registration produces a small error, computing the alignment as the accumulation of inter-frame registrations results in an accumulation of the error over time. This results in a bad alignment, which is especially important when the camera has close paths in its trajectory. Some techniques, like [75, 41], use a graphical model to deal with misalignments caused by the error accumulation. In these articles, nodes represent frames and edges represent adjacency in time or space. The mosaic image is generated by finding the optimal path that minimizes the alignment error. However, this approach is hardly applicable for real-time scenarios. First of all, complexity grows over time as new frames are added, while the minimization has to be repeated regularly. Secondly, if the mosaic has to be visualized, re-estimating means warping and interpolating the entire mosaic. Lastly, considering a reduced set of frames for lowering the computational cost might lead to not considering frames that are spatially but not temporally related.

Another possible approach is frame-to-mosaic alignment, which consists of registering the current frame directly with the mosaic image [74], or using instead a representation of the mosaic (like pixel mean or pixel median for each aligned frame). The method proposed in the previous chapter can be included in this category, where we represented the mosaic using a set of weighted histograms. This approach received some attention as a solution for preventing the error accumulation problem [88, 8]. Frame-to-mosaic alignment does not require prohibitive memory storage nor excessive computation time, but the alignment accuracy is affected. In this approach, either the frame has to be transformed to match with a mosaic region or vice versa. Small errors in the registration over the frame coordinates can be magnified when the registration is transformed into the mosaic coordinates. Similarly, using the mosaic coordinates is not a solution either. This effect is what we call precision degeneration (see section 4.2.2 for a more in-depth explanation). Furthermore, the computational drawback associated is twofold: the mosaic needs to be explicitly computed and the frame needs

to be warped before matching it with the mosaic. This also implies matching warped images obtained through pixel interpolation.

The remainder of this chapter is structured as follows: section 4.2 provides an intuitive overview of our method, explores the limitations of the related literature, and analyzes why our method can overcome them. Section 4.3 provides a more detailed description of the method, paying special attention to the DLIG alignment algorithm (section 4.3.2). Finally, in section 4.5 we detail the experiments conducted, including a set of quantitative and qualitative evaluations of the proposed algorithm.

4.2 Overview of the Proposed Method

4.2.1 Direct Local Indirect Global Alignment

We aim to achieve real-time video mosaicing. It is therefore natural to use the frame locality property to reduce the computational cost. Frame alignment is defined as a problem of local estimation -a similar approach to that of direct methods. But instead of computing the alignment of the whole image, which is computationally expensive, we compute the alignment of a discrete set of image subregions. Each of the subregions is matched using a tracking algorithm and an independent target model. Visual tracking is an important field of computer vision, and many robust and efficient algorithms have been proposed, a fact we take advantage of. However, the risk of errors during the tracking should be taken into account. For this reason we also impose global consistency on the set of estimations. In an ideal scenario, the local matchings should be consistent with a global image transformation. It should be possible to compute it as in indirect methods, and the point matchings should be consistent with the transformation. For this reason, we iteratively alternate the local alignment stage with a global transformation estimation. This last step is used to correct erroneous trackings and to impose global consistency with respect to the interframe transformation. This process yields robust consistent interframe matchings. We call this methodology Direct Local Indirect Global (DLIG) alignment method, since the local alignment is performed like in direct methods, while the global transformation is computed as in indirect methods.

4.2.2 Multi-frame Matching

Our method performs multi-frame matching as a way to prevent error accumulation problems. We show that it is capable of doing so while preventing the problems of frame-to-mosaic approaches. However, it is first necessary to provide a more in-depth explanation of the problems associated with frame-to-mosaic approaches.

We referred in the introduction to the problem of error accumulation in the frame-to-

frame alignment approach, and how multi-frame matching through the use of graphical models can make real-time applications unfeasible. Frame-to-mosaic approaches alleviate such a problem, but present instead other drawbacks. First of all, the mosaic has to be computed explicitly. Secondly, we match deformed images (i.e. interpolation). Last and most importantly, it suffers from what we called the precision degeneration problem. We now wish to give a more insightful explanation of this last problem and to show how to prevent all of these problems when performing multi-frame matching.

The precision degeneration problem happens when the coordinate system of the current frame i is very different from the reference coordinate system. In this case the computation of the alignment based on a set of matched points becomes unstable. If alignment T_i severely distorts the matched frame, then it also distorts the errors of the matching. Therefore, a small matching error in the mosaic can correspond to large matching errors in frame i or its reciprocal.

More formally, given two frame points so that $x = \bar{x} + \vec{\epsilon}$, their distance in frame coordinates is $\|\vec{\epsilon}\|$, while the distance between their mosaic equivalents is $\|T_i(\vec{\epsilon})\|$. Since T_i is not an Euclidean transformation, the same error $\|\vec{\epsilon}\|$ at different frame locations can yield (very) different errors in mosaic coordinates. Therefore, performing the estimation of alignment i in mosaic coordinates or in frame coordinates can yield very different results.

This effect is shown in Fig. 4.1. In this figure, two points, x'_1 and x'_2 , located in frame i are shown, together with a set of locations (circled) placed at a distance ϵ . When performing MSE estimation, the euclidean distance is equivalent to the error, so the circles represent points producing the same matching error. The problem arises when transforming these points of equal error into mosaic coordinates, since they are transformed into distorted ellipses.

In order to prevent these performance issues, we instead perform multi-frame matching. More precisely, we relate some locations typically belonging to different frames (called reference points for frame i) to a set of locations on the current frame i . The reference locations are selected using the mosaic coordinate system, so we can guarantee that they are seen in the current frame. We assume that we have an initial hypothesis of the current frame alignment \hat{T}_i^0 . Then, we can project the reference points into the mosaic coordinate system using their respective alignments, and then backproject them into the current frame through \hat{T}_i^0 . This generates a set of current frame locations, which are dependent on \hat{T}_i^0 . Each of these points is considered to be the center of an image patch, which will be used to perform the matching. At this stage, we can use the DLIG alignment to refine this first hypothesis into a final alignment. By following this methodology, the matching is performed using undistorted information from the frames to which the reference points belong. Since we do not use mosaic aspect information, it is not necessary to explicitly build the mosaic, deforming and interpolating patches for matching is avoided, and the precision degeneration problem is prevented.

It is important to note that this formulation is equivalent to considering the reference points to lay on the mosaic, since alignments of previous frames are fixed. It is just necessary

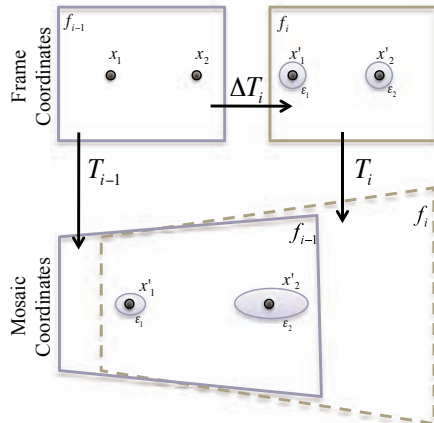


Figure 4.1: Precision degeneration. Euclidean distance is equivalent to point matching error. The circles drawn around points x'_1 and x'_2 show points at equal distance to their central locations. The interframe distortion is almost non-existent. In contrast, when projected onto the mosaic, they can suffer a great distortion, which hinders frame-to-mosaic alignment.

to take into account that the visual information used to perform their matching with points at the current frame will be extracted from their original frames. For simplicity, we will say these points lay on the mosaic.

As a final remark, note that this approach is only possible because we do not perform the frame matching typical of direct methods, but instead we split the frame information into parts that can be related to different frames. Since the mosaic efficiently summarizes the spatial information of the scene, it is natural to take advantage of it in order to obtain the frames spatially related to the current frame. But for the matching itself, we use the original frame information.

4.3 Method Description

Given a sequence formed by frames $\{f_i\}_{i=1:N}$, and a reference frame f_r , the aim is to compute for each frame f_i the alignment T_i that puts it in correspondence with the reference frame. We compute alignment T_i by using two sets of points put in correspondence, one from the current frame and another from the mosaic. In our work, T_i is restricted to be a 2D projective transformation, due to the good tradeoff provided between flexibility and simplicity on the alignment estimation. Thus, given a set of points $X' = \{x'_j\}$ in f_i coordinates and their corresponding points $X = \{x_j\}$ in the mosaic coordinates, we compute a transformation T_i that puts these

sets in correspondence (being in our case a 3x3 transformation matrix¹).

In the remainder of this section we will first describe how we estimate correspondences between frame points and mosaic points. Afterwards, the DLIG transformation estimation is described in detail. Then, a summary of our algorithm is provided.

4.3.1 Local Point Correspondences using a Tracking Algorithm

Each of the reference points is positioned with respect to a point on the current frame by using a template-based tracking algorithm. We consider each reference point to be the center of an image patch. Such a patch is used to extract a target representation (the model) to be used by the tracking algorithm. Then, this model is used to match the region against the current frame, and the center of the matched region is positioned with respect to the reference point. We use $I_k(x_j)$ to denote an image patch within frame k centered at point x_j .

More precisely, given a patch representation φ , matching a template within an image consists of finding the ideal Δp so that $\varphi(W(I_i, \Delta p))$ is the closest, under some criterion, to the model $\varphi(I_k(x_j))$. Here, Δp contains the parameters of a transformation W over I_i , defining the patch from where the representation is computed, for example, its center and size. It is important to note that the matching obtained depends on the type of transformation allowed. Δp can contain the parameters of, for example, a translation, an affine transformation, or a projective transformation. It is also important to note that at each iteration of a tracking algorithm, $\varphi(W(I, \Delta p))$ has to be computed. This generally implies computing the patch warp $W(I, \Delta p)$, and its computational cost depends on the tracking algorithm.

In practice, it is possible to obtain a model for the template matching from the mosaic image, and we would obtain a frame-to-mosaic alignment algorithm. However, to prevent the problems associated with this approach, we compute the model representation directly from frame information. More precisely, provided that x_j is a reference point in mosaic coordinates, we backproject it to its original frame, which we represent as $o(j)$, and compute the point representation using a subregion of $f_{o(j)}$ centered around $T_{o(j)}^{-1}(x_j)$. Therefore, the representation used is formally defined as:

$$\varphi(x_j) = \bar{\varphi} \left(T_{o(j)}^{-1}(x_j) \right) \quad (4.1)$$

where $\bar{\varphi}$ is an equivalent representation of φ applied to a different image, and we simplify the notation $\varphi(I(x_j))$ as $\varphi(x_j)$.

¹We use a 3×3 transformation matrix, since we only focus on 2D projective transformations. Nevertheless, the method can be extended to other types of transformations, such as polymorphic or piecewise affine transformations, despite the fact that they cannot be modeled using a 3x3 transformation matrix. Note that since the alignment includes translation, we use the homogeneous representation of points.

The representation $\bar{\phi}$ depends on the tracking algorithm used. For example, it can be the patch intensity values if we use correlation or a kernel-weighted histogram if kernel tracking is used. Some other tracking algorithms are flexible with respect to the representation used the representation used.

Each time a new part of the scene is observed, some new reference points are added to the set of reference points and their representation is computed. The definition of the set of reference points is incremental, which means that it is possible to add more, but those that have been added are always kept.

Since the information stored is just a model representation for each reference point (typically a vector), the memory storage benefit of this approach is important with respect to approaches requiring storage of the whole mosaic. Furthermore, at each new frame we are using previously computed representations, so we do not need to recompute them at each step, speeding up the process.

Finally, it is important to remark that the properties of the tracking algorithm used are inherited by the mosaic. For example, a tracking algorithm incapable of coping with varying illumination will fail in such cases, while using a representation robust to illumination changes will boost the performance. We will provide some examples in section 4.5, where we show the performance of different tracking algorithms.

4.3.2 Direct Local Indirect Global Alignment Computation

We now describe the computation of alignment T_i using the DLIG algorithm. We need to find, and put in correspondence, two sets of reference points: X_i , belonging to the mosaic; and a set X'_i , belonging to frame i .

We rely on the region matching procedure described in the previous section. However, we impose global consistency over the estimated matchings to prevent local mismatches that might arise from tracking errors, or from the content of the current frame, for example, when moving objects are present. Algorithm 1 summarizes this process, and its details are provided below.

We perform an iterative estimation of T_i , starting with $\hat{T}_i^0 = T_{i-1}$ as the initial guess². Given the set of mosaic points X_i , the first step consists of projecting them onto the current frame coordinate system using the current estimate of the alignment, \hat{T}_i^0 , obtaining a set of points $\bar{X}_i^0 = (\hat{T}_i^0)^{-1} X_i$ (line 3 of algorithm 1). Due to the frame locality property, the points \bar{X}_i^0 are close to the as-yet-unknown X'_i , which we aim to estimate.

²It is possible to add a dynamic model to predict the evolution of T_i and obtain a better initial guess.

The tracking algorithm is then applied to the locations \bar{X}_i^0 . However, instead of sequentially performing a set of iterations, we just apply one (or a small predefined set of) iteration of the tracking algorithm. Through this step, a new estimate of X_i^t is obtained, denoted by \hat{X}_i^0 (algorithm 1, line 4). Furthermore, it is possible to obtain the error associated with each one of the estimates inherently given by the tracking algorithm. The particular form of such error depends on the tracking algorithm. For example, it is possible to use the sum of square differences between the target model and the representation of the estimated region.

We then combine the Direct Local step with an Indirect Global step. More precisely, the Indirect Global step consists of obtaining a new estimate of alignment i by using the point estimates provided by the Direct Local step. The transformation estimate is obtained by weighting each of the local estimates (algorithm 1 line 5). We will define how to compute the weights W_i in the following section, although the errors associated with each point matching play an important role in such a definition. The weighted estimation is achieved by using Weighted Mean Square Error minimization. The resulting alignment estimation is denoted by \hat{T}_i^1 . By adding the Indirect Global step, the next iteration of the Direct Local step will be performed over the set of points $\bar{X}_i^1 = (\hat{T}_i^1)^{-1} X_i$ instead of the points \hat{X}_i^0 . In this way, the correctly estimated locations help computing the alignment estimate, and the alignment estimate is used to back-projecting the mosaic points onto the current frame, correcting possible errors obtained on the Direct Local step.

Algorithm 1: DLIG Alignment Computation

```

1  $\hat{T}_i^0 = T_{i-1}$ ;
2 for  $t \leftarrow 0$  to Until Convergence do
3    $\bar{X}_i^t = (\hat{T}_i^t)^{-1} X_i$ ;
4    $\bar{X}_i^t \rightarrow \hat{X}_i^t$  using a tracking step;
5    $\hat{X}_i^t \rightarrow W_i^t$ ;
6    $(\hat{X}_i^t, W_i^t) \rightarrow \hat{T}_i^{t+1}$  using WMSE;
7  $T_i = \hat{T}_i^{end}$ ;
```

Fig. 4.2 contains an example showing the robustness of our method to moving objects and a changing background. In this sequence, the regions to be matched with moving objects yield high errors and are correctly assigned with a low weight when computing the alignment.

4.3.3 Mosaic Construction using the DLIG Algorithm

Here we provide the remaining details of our algorithm. First of all, the mosaic is initialized as the first frame of the sequence. Therefore, the first alignment is the identity, $T_1 = I_3$. Afterwards, a set of reference points are created using f_1 at locations placed over a regular lattice. This means a set of locations $\{x_j\}_{j=1:N_1}$ from the mosaic are selected, and their representations $\phi(x_j)$ are computed.



Figure 4.2: Green squares are patches centered at \hat{X}_i^j . The intensity of the color reflects the weights used by the WMSE. Those regions over the moving car have consistently low weight and therefore have a low impact on the alignment computation. The upper images are 4 consecutive frames; the lower image represents the corresponding mosaic.

For frame i , a subset of the reference points $\{x_j^i\}_{j=1:N_i}$ are selected. Alignment $i - 1$ is used to select them, and to initialize the computation of alignment i . Thus, when projected onto frame i , the selected reference points produce a set $\{x_j^i\}_{j=1:N_i} = \{T_{i-1}^{-1}(x_j^i)\}_{j=1:N_i}$. Now, the DLIG alignment is computed as described by algorithm 1.

In our case, we use two criteria to define the weights used. The first one is the error yielded by the Direct Local step, denoted by $\{\varepsilon_j\}$, and its particular form depends on the type of tracking algorithm used. The second criterion is based on the number of times the reference point has been successfully used, denoted by as τ_j . The resulting form of their computation is:

$$w_j = C_2 \cdot \frac{\tau_j \cdot \eta_j}{\sum_k \tau_k \sum_k \eta_k} \quad (4.2)$$

where $\eta_j = C_1 \cdot e^{-\frac{N\varepsilon_j}{\sum_k \varepsilon_k}}$

where C_1 and C_2 are normalization factors. This definition of the weights makes the estimation robust with respect to moving objects in the scene, reflected by a high error (implying low η_j) and to unstable image regions, reflected in the value of τ_j .

This process is iterated until convergence. Again, any convergence criterion can be used. We set two different criteria. The first one consists of a maximum number of iterations, imposed to prevent excessive computational cost. The lack of convergence can be produced by poor estimates of the DLIG algorithm, but also due to natural image situations such as moving objects or regions with low texture. We impose a second criterion based on the lack of change in successive iterations, which is a typical convergence criterion.

This process finally results in the estimation of alignment T_i . Once T_i is computed, we update all the reference point models in order to adapt them to the new conditions of the video (e.g. changes in illumination). Any updating strategy can be used. We opt for updating the model as a linear combination of the current model and the performed estimation [55]. Furthermore, we compute the variance of the model in all the cases it was observed in a frame. This value can be used to lower the weight of badly conditioned reference points, for example, those placed in a region with a varying aspect (e.g. a tree, a part of the image with a lot of movement, etc.).

The pseudocode of our mosaicing algorithm using DLIG alignment is presented in algorithm 2.

Algorithm 2: Mosaic construction using DLIG alignment. RP stands for reference points

Data: Video Sequence $F = \{f_i\}_{i=1:N}$
Result: Mosaic image M

```

1 begin
2   •  $T_1 = I_3; RP = \emptyset; M = f_1;$ 
3   •  $RP \leftarrow$  Generate RP using  $f_1;$ 
4   for  $i \leftarrow 2$  to  $N$  do
5     •  $RP' \leftarrow$  Select a subset of RP fitted by  $T_{i-1};$ 
6     • DLIG alignment as depicted in algorithm 1;
7     • Update the  $RP'$  models using  $f_i;$ 
8     • Update the mosaic  $M$  using  $f_i;$ 
9     •  $RP \leftarrow$  Add new RPs from  $f_i$  if they belong to new parts of the mosaic;

```

4.4 Description of the tracking algorithms considered in this work

The different tracking algorithms considered to perform the direct local step are the following: Normalized Cross Correlation (NCC) based tracking [48], Lucas-Kanade (LK) [52, 4], SIFT [13], and Kernel Tracking (KT) [33]. They are selected to be representative of different approaches to tracking and for having been applied to already-existing mosaicing algorithms. NCC is used as a comparison baseline. LK is the basis of a large number of mosaicing algorithms, and it is a widely extended tracking paradigms as well. In the case of SIFT, despite not being by nature a tracking algorithm, it can be used to find correspondences between frames. We include it since SIFT features have shown to be very effective when applied to mosaicing. Finally, KT has some properties which make it especially suited for mosaicing purposes. It is robust to small changes in the target aspect, as those produced by lightning and small target deformations, and it is computationally very efficient.

We do not describe Kernel Tracking in this chapter, since a detailed description can be found in chapter 3.

4.4.1 Normalized Cross-Correlation

Cross Correlation is an exhaustive template matching algorithm based on minimizing the squared Euclidean distance between pixels.

The squared Euclidean distance is defined as:

$$d_{f,t}^2(u, v) = \sum_{x,y} [f(x, y) - t(x - u, y - v)]^2 \quad (4.3)$$

where the sum is over x, y ; f is an image; and t is a template positioned at u, v . By expanding d^2 , we get:

$$d_{f,t}^2(u, v) = \sum_{x,y} [f^2(x, y) - 2f(x, y)t(x - u, y - v) + t^2(x - u, y - v)^2] \quad (4.4)$$

Note that $\sum_{x,y} t^2(x - u, y - v)$ is constant and if we consider $\sum_{x,y} f^2(x, y)$ to be approximately constant, then we can simplify the equation to a more efficient form as:

$$c(u, v) = \sum_{x,y} f(x, y)t(x - u, y - v) \quad (4.5)$$

where, in this case, our goal is to find the location that maximizes this similarity function. This similarity function has several disadvantages: $\sum_{x,y} f^2(x, y)$ is not constant and varies with position and the range of $c(u, v)$ varies depending on the size of the template and it is not robust to changes in light conditions.

Normalized Cross Correlation (NCC) [48] overcomes these drawbacks by normalizing the image and the template to the unit length, giving us a cosine-like correlation coefficient, defined as:

$$\gamma(u, v) = \frac{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}][t(x - u, y - v) - \bar{t}]}{\sqrt{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}]^2 \sum_{x,y} [t(x - u, y - v) - \bar{t}]^2}} \quad (4.6)$$

where \bar{t} is the template pixel mean and $\bar{f}_{u,v}$ is the frame mean of the region under the feature.

4.4.2 Lucas Kanade

There exist multiple versions of the Lucas Kanade algorithms. In this work we have implemented the most common ones, in order to determine the best approach for the DLIG alignment. The different four algorithm implemented are: Lucas Kanade Addive/Compositional and Lucas Kanade Inverse Additive/Compositional. In addition, for each algorithm, we have implemented two version, one for estimating only translations and another for estimating affine transformations.

In Fig. 4.3 and 4.4 show the comparison of the different algorithms proposed. The presented results, were obtained by tracking 8000 points on 10 different datasets of 10 synthetic images each. Fig. 4.3 shows the error histogram based on Euclidean distances (in pixels) with

respect to the algorithm output and a ground truth. Besides, fig. 4.4, shows the histogram of iterations needed to converge for each point. Note that the values are shown in percentages in order to simplify the evaluation.

In the next subsections we describe in detail the Lucas Kanade methods implemented in this work.

Lucas Kanade Additive

Lucas Kanade Additive, presented in [52], was the first tracking algorithm of this kind. In this work, we follow the notation and description presented in [4]. The goal of the Lucas Kanade algorithm is to minimize the sum of squared error between two images, a template T and an image I warped back onto the coordinate frame of the template:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2, \quad (4.7)$$

where $\mathbf{x} = (x, y)^T$ is the pixel coordinates and $\mathbf{W}(\mathbf{x}; \mathbf{p})$ is the warping function (which could be a transformation matrix) applied in the point \mathbf{x} with parameters \mathbf{p} .

Lucas Kanade assumes that \mathbf{p} is known and iteratively minimizes equation 4.7, for increments of $\Delta \mathbf{p}$:

$$\mathbf{p} = \mathbf{p} + \Delta \mathbf{p}. \quad (4.8)$$

and equation 4.7 becomes:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta \mathbf{p})) - T(\mathbf{x})]^2, \quad (4.9)$$

The Lucas Kanade algorithm uses Taylor series to optimize the non-linear expression in equation 4.9. By performing a first order Taylor expansion on the warping function, we get:

$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x})]^2, \quad (4.10)$$

where $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ is the *image gradient* evaluated at $\mathbf{W}(\mathbf{x}; \mathbf{p})$ and $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the *Jacobian* of the warp, computed as:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial \mathbf{W}_x}{\partial p_1} & \frac{\partial \mathbf{W}_x}{\partial p_2} & \dots & \frac{\partial \mathbf{W}_x}{\partial p_n} \\ \frac{\partial \mathbf{W}_y}{\partial p_1} & \frac{\partial \mathbf{W}_y}{\partial p_2} & \dots & \frac{\partial \mathbf{W}_y}{\partial p_n} \end{pmatrix}. \quad (4.11)$$

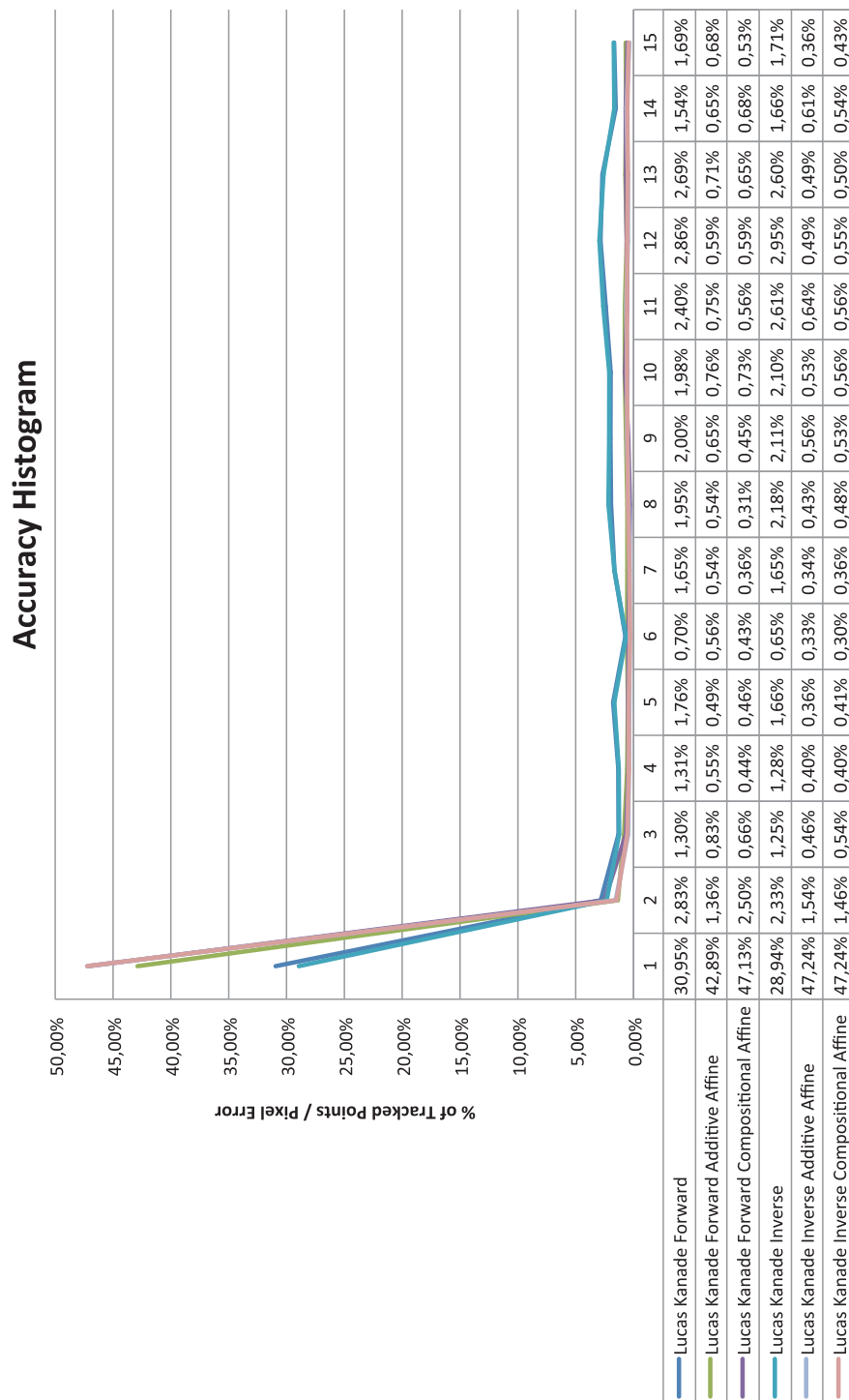


Figure 4.3: Histogram of error produced (Euclidean distance in pixels) with respect to the algorithm output and a ground truth. This graphic was made by tracking 8000 points in ten different synthetic images.

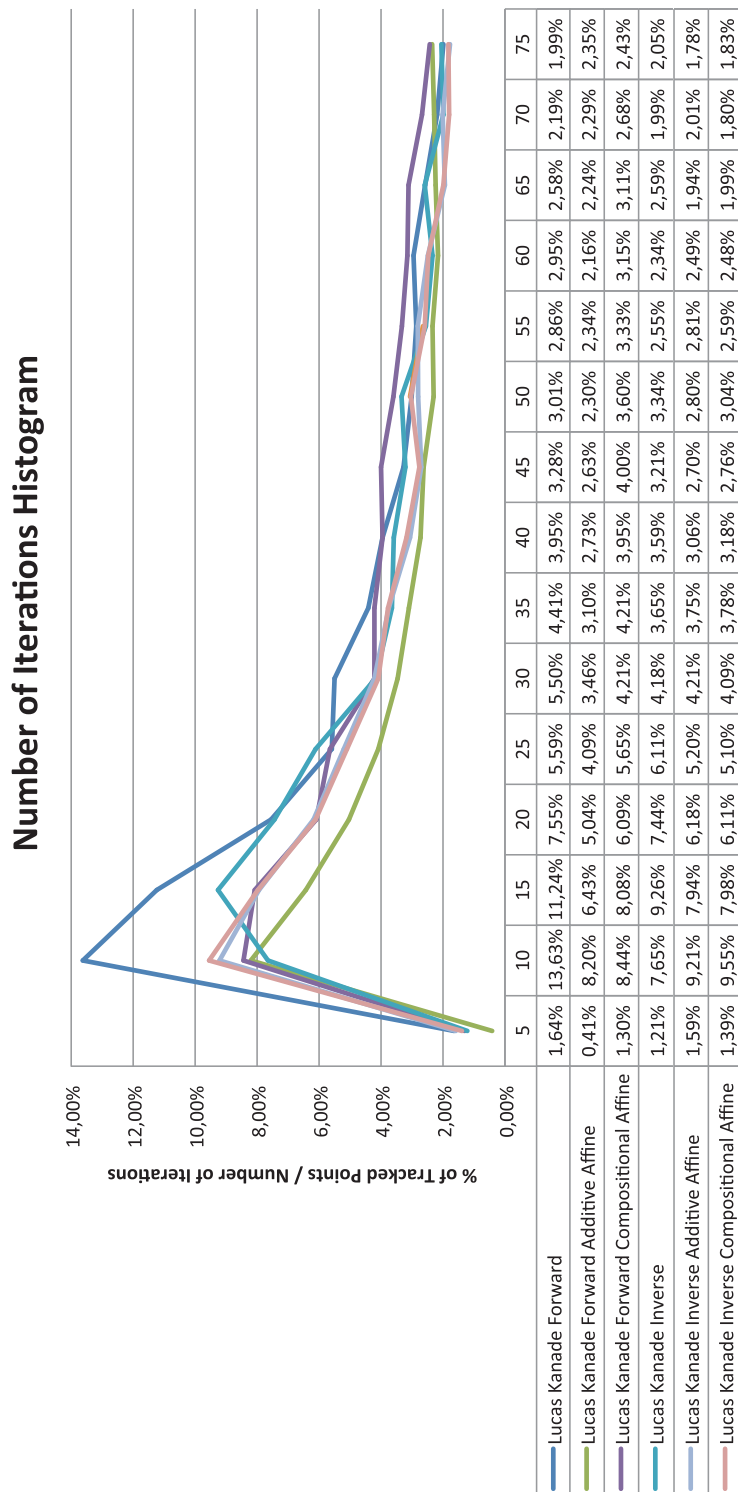


Figure 4.4: Histogram of iterations needed to converge for each Lucas Kanade-based algorithm. The results were obtained by tracking 8000 point in ten different synthetic images.

where n is the number of parameters of our warping model. In this case \mathbf{p} has only 2 parameters, translation on x and y , giving us:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \quad (4.12)$$

Setting equation 4.10 equal to zero and solving gives the closed form solution:

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]^T [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))], \quad (4.13)$$

where H is the $n \times n$ (Gauss-Newton approximation to the) *Hessian* matrix:

$$H = \sum_{\mathbf{x}} [I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]. \quad (4.14)$$

which is in this case is a 2×2 matrix.

The affine version of this algorithm only differs by the warping function. In this case, it has six parameters corresponding to Affine transformations. Then $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ becomes:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \quad (4.15)$$

and the Hessian H is now a 6×6 matrix. The rest of the algorithm is the same.

The Lucas Kanade Additive algorithm is less demanding in terms of computation time with respect to the NCC. However, Lucas Kanade Additive needs an accurate initialization to find a proper solution. If the misalignment between the template and the image is bigger than a half of the size of the template, finding a good solution becomes impossible. In addition, this algorithm requires a lot of iterations to converge.

Lucas Kanade Compositional

In this case the alignment proposed in sec. 4.4.2, is improved by updating the warping function using the *compositional* method proposed in [76]. In this case, the warping function is:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p}) \equiv \mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p}); \mathbf{p}). \quad (4.16)$$

and can be computed easily in a matrix form,

$$\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p}); \mathbf{p}) = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1+\Delta p_1 & \Delta p_3 & \Delta p_5 \\ \Delta p_2 & 1+\Delta p_4 & \Delta p_6 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (4.17)$$

Note that when estimating only translations the additive and compositional methods are equivalent. The compositional version reduces the number of iterations in the Lucas Kanade algorithm.

Lucas Kanade Inverse Additive

Lucas Kanade Inverse Additive was presented in [32]. This algorithm increase the efficiency by switching the role of the image and the template. In this case, the inverse compositional algorithm minimizes:

$$\sum_{\mathbf{x}} [T(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]^2, \quad (4.18)$$

with respect to $\Delta \mathbf{p}$ and then updates the warp:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \mathbf{W}(\mathbf{x}; \mathbf{p}) - \mathbf{W}(\mathbf{x}; \Delta \mathbf{p}). \quad (4.19)$$

After performing first order Taylor expansion on equation 4.18, this gives:

$$\sum_{\mathbf{x}} [T(\mathbf{W}(\mathbf{x}; 0)) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]^2, \quad (4.20)$$

where $T(\mathbf{W}(\mathbf{x}; 0))$ is the identity warp. Finally, setting equation 4.20 to zero and solving gives the closed form solution:

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})], \quad (4.21)$$

where the Hessian H is in this case defined as:

$$H = \sum_{\mathbf{x}} [T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T [T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]. \quad (4.22)$$

In this approach the gradient, the Jacobian and the Hessian can be precomputed since they do not depend on \mathbf{p} . Therefore, this approach is very convenient for improving the overall

computation time.

The affine version only differs in the Jacobian function and it is computed as in equation 4.15.

The rest of the alignment algorithm is the same as the one presented in sec. 4.4.2.

Lucas Kanade Inverse Compositional

This version of the Lucas Kanade algorithm was presented in [23, 3]. In this case, the parameters are updated using the compositional method:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1} \equiv \mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}; \mathbf{p}). \quad (4.23)$$

the only difference from the update in the forwards compositional algorithm in equation 4.15 is that the incremental warp is inverted before it is composed with the current estimate. It can be computed easily in a matrix form:

$$\begin{aligned} \mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}; \mathbf{p}) = \\ \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1+\Delta p_1 & \Delta p_3 & \Delta p_5 \\ \Delta p_2 & 1+\Delta p_4 & \Delta p_6 \\ 0 & 0 & 1 \end{pmatrix}^{-1} - \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \quad (4.24)$$

Again we want to remark that when the warping model only covers translations the compositional and additive versions are equivalent.

We have used the results obtained with this approach to present the experimental results in the next section, since the accuracy is similar to Lucas Kanade Affine Additive with a notable reduction in computational time and number of iterations.

Feature method based on SIFT

Feature based methods find the point correspondences by first selecting a set of locations in both images. The points are selected to have representative local image properties so they are likely to be detected independently of the point of view of the camera. Then for each point a descriptor is extracted, which could be color, gradients, etc. Finally, the transformation is computed as the one maximizing the matching between both the set of locations over the current frame and the set of locations over either another frame or the mosaic. In this work we use a similar approach to [13, 85]. Both extract a set of SIFT features from all the images

and then use algorithms for eliminating outliers. In our case we use RANSAC. We have used the OpenCV implementation of the SIFT algorithm.

4.5 Experimental Results

In this section we have 3 main aims. The first one is to evaluate the performance of our method using different types of tracking algorithms. Secondly, we deal with the performance of the DLIG alignment with frame-to-frame and frame-to-mosaic alignment strategies. Thirdly, we select the best performing tracking algorithm and show some performance examples. We include in our evaluation interlaced videos and changes in illumination, since they are very common in practical applications.

4.5.1 Description of the test videos

In order to evaluate the performance of the different mosaicing algorithms, we generated a set of synthetic video sequences from large real images, simulating different camera movements. We use synthetically created videos because it is otherwise very difficult to obtain a ground truth allowing a quantitative performance evaluation. More precisely, these videos are generated by applying a transformation pattern to 2363x1073 static images of different characteristics and then cropping a 640x480 patch from the center of the transformed image. We have generated two different transformation patterns. The first one is defined by equation 4.25. It contains a projective warp and a translation.

$$T_i = \begin{pmatrix} 1 & 0 & -10i \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha_i) & 0 & \sin(\alpha_i) \\ 0 & 1 & 0 \\ -\sin(\alpha_i) & 0 & \cos(\alpha_i) \end{pmatrix} \quad (4.25)$$

where $\alpha_i = \frac{-0.001\pi i}{180}$. The second pattern is defined by equation 4.26. It contains an affine transformation, including scaling, rotation and translation.

$$T_i = \begin{pmatrix} 0.99^i \cos(\beta_i) & -0.99^i \sin(\beta_i) & 5i \\ 0.99^i \sin(\beta_i) & 0.99^i \cos(\beta_i) & 5i \\ 0 & 0 & 1 \end{pmatrix} \quad (4.26)$$

where $\beta_i = \frac{i\pi}{180}$.

The final sequences consist of 35 frames obtained by incrementally applying the transformation patterns and another 35 frames consisting of the reverse of the first 35 frames. This results in 70 frames per sequence, where the first frame and the last frame match. Fig. 4.5 shows a scheme of how these videos are generated.

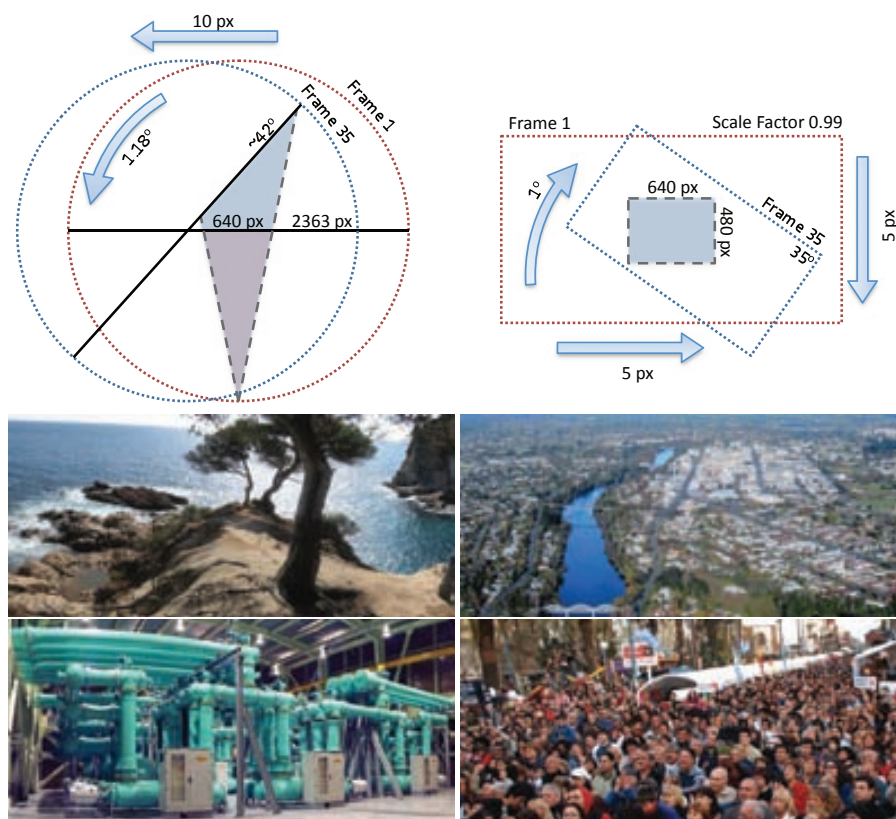


Figure 4.5: The top half shows a scheme of how the video sequences are generated (projective pattern on the left, affine one on the right). On the bottom, the images used to generate the 16 test sequences.

Additionally, we created an interlaced version of each of these sequences in order to evaluate the algorithm performance in such conditions. It is important to note that the minimum error to expect in interlaced videos is dependent on the camera movement. The first frame of the sequences is not interlaced and when compared to the rest of frames, which are already interlaced, an error is naturally produced and propagated over all the process (our error measure is an average of pixel position errors). Therefore, the objective in interlaced videos is not to have no error but instead to not accumulate further error throughout the sequence. Fig. 4.6 shows a comparison between the progressive and the interlaced version of a frame. In total we have 16 different video sequences for quantitative testing purposes.

4.5.2 Implementation details and results obtained

Each test is initialized using a set of reference points placed on a regular 7x5 lattice, using patches of 70x70 pixels to construct their representation.

The results of comparing the different tracking algorithms considered, together with a performance comparison with respect to the different alignment approaches, are shown in table 4.1. To quantitatively evaluate the mosaicing error, we compute for each pixel the Euclidean distance between its location through the estimated transformation and its real location. We provide the mean and variance of these errors in the first columns. This table also includes graphics summarizing the evolution of the error along the sequence; for example it is possible to see the error accumulation phenomenon for the case of frame-to-frame alignment. These graphics are composed of 70 concatenated vertical histograms (one per frame), where the x-axis represents frame number (from 1 to 70) and the y-axis shows the error in pixel distance (from 0 to 20). Each of these histograms is constructed by considering the error for each pixel in the corresponding frame for all the test videos. Darker tones of a bin represent high density of pixels having an error corresponding to this bin. We also show over the graphics the mean error using red dots. Note that the errors computed refer to the accumulated estimations and do not refer to interframe errors.

It is possible to see from the evaluation that the performance of the DLIG alignment is consistently superior. One important aspect is that the error is not accumulated over the sequences. In contrast, the frame-to-frame approach presents consistent error accumulation problems. For the case of frame-to-mosaic approaches, NCC and LK perform well, but the error again increases with the sequence. For the case of frame-to-mosaic, we use a frame-to-mean-mosaic approach.

		Progressive Videos				
Alignment	Tracker	F.T.	μ px	σ px	Error/Frame D.	
Frame to Frame	NCC	–	3.555	2.406		
	LK	–	25.94	15.85		
	SR	1	12.10	17.05		
	KT	–	2.781	2.170		
Frame to Mosaic	NCC	–	3.162	2.180		
	LK	2	4.794	2.367		
	SR	6	–	–		
	KT	–	225.5	195.5		
DLIG Alignment	NCC	–	1.486	0.944		
	LK	–	3.952	1.8849		
	SR	–	1.240	0.841		
	KT	–	1.186	0.810		

Table 4.1: Alignment error comparison. SR stands for SIFT + RANSAC and KT for Kernel Tracking. The graphics show in the x -axis the frame number, from 1 to 70, and on the y -axis (ranging from 0 to 20) the computed error in terms of mean for the frame pixels (red) and their quantized distribution (grey). See text for details on the graphic construction. Error/Frame D. means Error/Frame Distribution.

		Interlaced Videos			
Alignment	Tracker	F.T.	μ px	σ px	Error/Frame D.
Frame to Frame	NCC	1	18.78	27.10	
	LK	1	26.52	14.97	
	SR	8	–	–	
	KT	8	–	–	
Frame to Mosaic	NCC	8	–	–	
	LK	8	–	–	
	SR	8	–	–	
	KT	8	–	–	
DLIG Alignment	NCC	1	5.532	3.459	
	LK	–	6.259	3.582	
	SR	8	–	–	
	KT	–	4.646	1.386	

Table 4.2: Alignment error comparison. SR stands for SIFT + RANSAC and KT for Kernel Tracking. The graphics show in the x -axis the frame number, from 1 to 70, and on the y -axis (ranging from 0 to 20) the computed error in terms of mean for the frame pixels (red) and their quantized distribution (grey). See text for details on the graphic construction. Error/Frame D. means Error/Frame Distribution.



Figure 4.6: A progressive frame from a test sequence (left), and the same frame of the interlaced version of the video sequence (right).

Since the DLIG alignment relies on the tracking performance, it is safe to say that the final mosaicing algorithm inherits the properties and limitations of the tracking algorithms. The most illustrative case is the performance with interlaced videos for SIFT and kernel tracking. Whereas they perform very similarly for deinterlaced videos, SIFT completely fails with the interlaced ones. This is due to the nature of the SIFT descriptor, based on orientation of edges, which are completely distorted in interlaced videos. In contrast, kernel tracking performs very well with interlaced videos. The representation used in kernel tracking, kernel-weighted histograms, is well known for being very robust against blurring. In practice, for kernel tracking the representations of the interlaced and deinterlaced version of the same region are very similar.

Given the results of the performance evaluation, we adopt the kernel tracking as the most reliable, both in interlaced and progressive videos. It is also the less computationally demanding.

We made some further performance evaluations to establish the optimal parameters of the representation for kernel tracking. More precisely, we selected the optimal number of kernels and whether isotropic or anisotropic kernels should be used. We tested kernel-weighted representations formed by 1, 4 and 9 kernels (see Fig. 4.7). The experimental results are shown for progressive videos in table 4.3 and for interlaced videos in table 4.4, where is highlighted in bold the best configuration. It can be seen that there is an improvement in the performance between using 4 kernels compared to using just 1. Using 9 kernels leads to almost no performance improvement, while the computational cost grows with the number of kernels. The use of anisotropic kernels further improves the performance (1.186 vs. 1.015 average error respectively). In order to match the same scene region seen in two different frames, it would be necessary to consider projective transformations. Therefore, the kernels responsible for creating the patch representation should undergo the same transformation, leading to anisotropic kernels. However, it is possible to approximate them using isotropic kernels, as depicted in Fig. 4.7. It is therefore reasonable that the obtained results are less accurate. But

Tracker	Progressive Videos		
	μ (px)	σ (px)	Error/Frame Distribution
Kernel Tracking with 1 Isotropic Kernel	1.3952	1.1139	
Kernel Tracking with 4 Isotropic Kernels	1.1864	0.8101	
Kernel Tracking with 9 Isotropic Kernels	1.1384	0.72221	
Kernel Tracking with 1 Anisotropic Kernel	1.1258	0.94817	
Kernel Tracking with 4 Anisotropic Kernels	1.0148	0.59686	
Kernel Tracking with 9 Anisotropic Kernels	0.98805	0.54936	

Table 4.3: DLIG alignment accuracy depending on the Kernels Tracking configuration on progressive videos

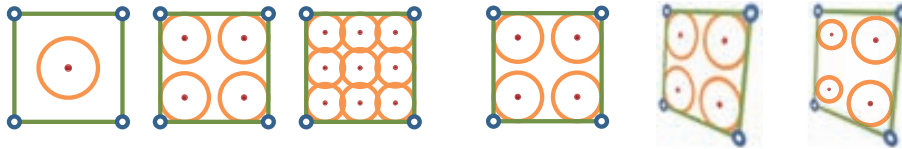


Figure 4.7: Left images: Region representation using different number of kernels (1, 4 or 9). Right: a projective deformation of a region produces anisotropic kernels. It can be approximated using isotropic kernels

anisotropic kernels present a computational drawback, since the growth in computational cost is significant. When using isotropic kernels instead, it is possible to generate look-up tables. A look-up table is a large set of precomputed kernels at different scales, so whenever a kernel suffers a scaling, it is only necessary to look for it on the table instead of computing its values again. This is possible when the kernels are isotropic, since only the scale can change. For anisotropic kernels instead, the variability is too high, since an anisotropic kernel is defined by 3 parameters instead of the 1 required for isotropic kernel. If the computational cost is not a restriction, the use of anisotropic kernels would fit best. In our case we use isotropic kernels because of the real-time requirement.

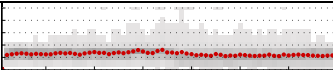
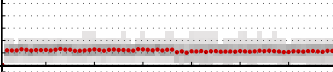
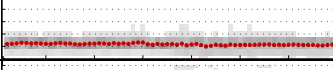
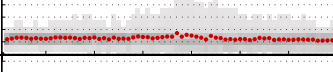
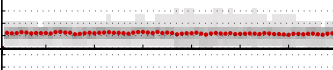
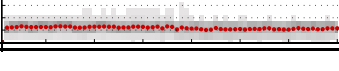
Tracker	Interlaced Videos		
	μ (px)	σ (px)	Error/Frame Distribution
Kernel Tracking with 1 Isotropic Kernel	5.0203	2.1828	
Kernel Tracking with 4 Isotropic Kernels	4.6457	1.3862	
Kernel Tracking with 9 Isotropic Kernels	4.6198	1.3052	
Kernel Tracking with 1 Anisotropic Kernel	5.0694	2.3921	
Kernel Tracking with 4 Anisotropic Kernels	4.6938	1.52	
Kernel Tracking with 9 Anisotropic Kernels	4.6414	1.3439	

Table 4.4: DLIG alignment accuracy depending on the Kernel Tracking configuration on interlaced videos

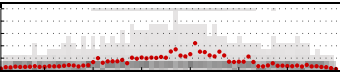
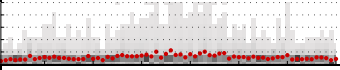
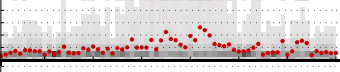
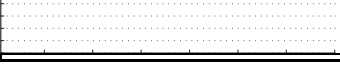
I.V. Range	F	Global				Error/Frame
		I.V.		Error		
		μ	σ	μ	σ	
[-5,5]	–	2.47	1.46	2.77	3.79	
[-10,10]	4	5.15	2.82	2.33	2.43	
[-15,15]	3	7.54	4.27	3.89	5.46	
[-20,20]	6	–	–	11.1	14.9	

Table 4.5: Kernel Tracking DLIG mosaicing accuracy under two illumination change patterns. I.V. stands for intensity variation. F stands for failed mosaics. The graphics are constructed as in table 4.1 (see text for further explanation)

I.V. Range	Block					Error/Frame
	F	I.V.		Error		
		μ	σ	μ	σ	
[-5,5]	–	2.48	1.44	1.65	1.60	
[-10,10]	2	5.00	2.88	2.19	2.28	
[-15,15]	2	7.52	4.30	3.72	4.25	
[-20,20]	3	10.1	5.75	5.50	7.11	

Table 4.6: Kernel Tracking DLIG mosaicing accuracy under two illumination change patterns. I.V. stands for intensity variation. F stands for failed mosaics. The graphics are constructed like in table 4.1 (see text for further explanation)

In order to test the capability of the DLIG algorithm to resist illumination changes, we designed a modification of the 8 synthetic test videos described before. For each of these frames, a random illumination variation pattern was added. The results of this experiment are shown in tables 4.5 and 4.6. We performed two different experiments depending on the illumination pattern added (numbers refer to a scale of $[0, 255]$). In the first one, uniform global noise was added to the frame, where the value of the illumination change was randomly selected within an interval following a uniform distribution. These results are shown on the left part of the table. It is important to note that having an interval of $[-10, 10]$ means the illumination change between two consecutive frames might be up to 20. The second pattern used divided the image into 16 blocks, and each of them was transformed using a uniform illumination pattern, again with a value randomly selected using a uniform distribution within a range. It is worth mentioning that these tests are performed without applying any illumination normalization to the frames, which would be recommended for the final application.

It is possible to see that for the block illumination pattern the results are better than for the global illumination one. Since the intensity of the variation is random, it is expected that some of the 16 blocks have a smaller illumination change. In these cases, the alignment can rely more on such regions.

The robustness of the alignment method to illumination changes depends on the properties of the tracking algorithm. For these experiments we used 12-bin kernel-weighted histograms. In practice, the bin width is of about 20. This implies robustness against illumination changes smaller than the width of the bin (like for those changes within the $[-5, 5]$ range), while for comparable or larger illumination changes the matching becomes unstable.

We also tested the performance of our method on a large sequence with slow changes



Figure 4.8: Comparison of light changes in a long interlaced sequence recorded using a Sony handycam DCR-PC110E. The blocks building the image correspond to frame 150, frame 37000, and their difference. It is important to note the color change, which produces some differences when subtracting the frames.

in the illumination, so the performance depends to a large degree on the ability to adapt the model to the ongoing changes. In Fig. 4.8, we combine blocks of frame 150, frame 37000, and their difference. It is possible to see the precision of the matching performed, even when there are 36850 frames in-between, which at 25 fps adds more than 25 minutes. Again, no illumination normalization was used in this sequence.

In order to provide a qualitative vision of the performance of our method, we show some experiments obtained with real sequences. The first experiment shows that our method does not accumulate error when building a mosaic from a large sequence. The obtained result is shown in Fig. 4.9. We used a sequence containing a typical surveillance situation, where the camera rasterizes a scene, with mostly but not only panning movement. The trajectory of the camera is shown over the mosaic. The sequence is composed by 2200 interlaced frames recorded using a Sony handycam DCR-PC110E with a resolution of 640x480 pixels, and the video is interlaced. The experiment consists of showing regions of different frames which share the same mosaic coordinates. If the mosaic is perfect these regions should be equal (except for illumination changes or moving objects). For example, frames 733, 979 and 1762 have a significant overlap, and the difference between the overlapping parts is shown. The same is done with frames 407, 1223 and 1367. The difference between these frames is small, even taking into account that the video is interlaced. Only regions with moving objects show a significant difference.

Some more qualitative examples are shown in Fig. 4.10 and 4.11. The figure shows the mosaics obtained and summarizes the field of view evolution within the sequences.

4.5.3 Memory requirements and computational cost

Our algorithm is capable of performing real-time video mosaicing, considering that the mosaic is not explicitly computed nor visualized. In order to achieve real time also for these functionalities, it is necessary to use multithreaded programming. For example, [51] achieve it by using CUDA language and two parallel threads.

We have built a C++ single-threaded implementation of this algorithm capable of running in real time for videos with high resolution. For example, we are capable of processing a 720x512 resolution video, excluding frame load, at 48,04 frames per second using a PC with a Core i7 920 (2,66Ghz) CPU, or at 28,98 frames per second on a laptop with a Core 2 Duo T7200 (2Ghz) CPU. Another important property is that the computational cost remains constant with time.

The memory storage requirements are very small. The amount of stored data grows linearly with respect to the extent of the mosaic, since only the representation of the reference points has to be stored, with each of these representations being just a vector.

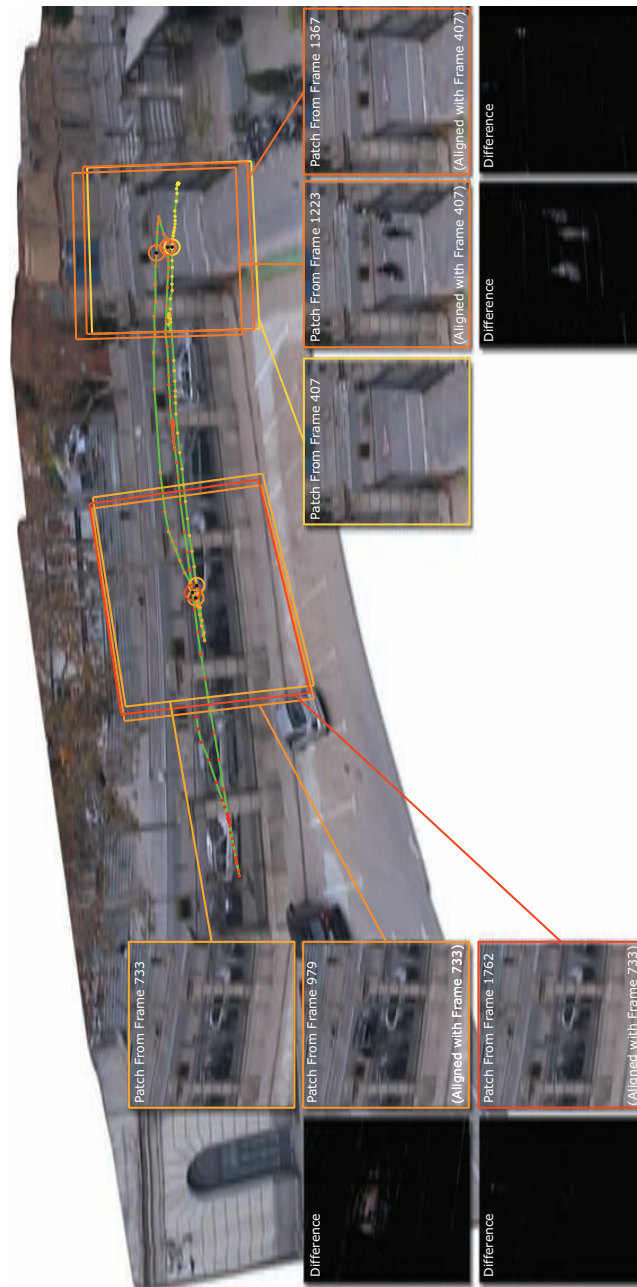


Figure 4.9: The figure shows the mosaic generated with the entire sequence (2200 frames). The green line shows the camera's trajectory and the dots mark the point of view each 10 frames. The color code indicates the frame, where the yellow color is used for the first frames, and the red ones for the last. The comparisons result from subtracting frame 733 from 979, and from 1762 (two differences on the left) and 407 from 1223 and 1367 (the two on the right).

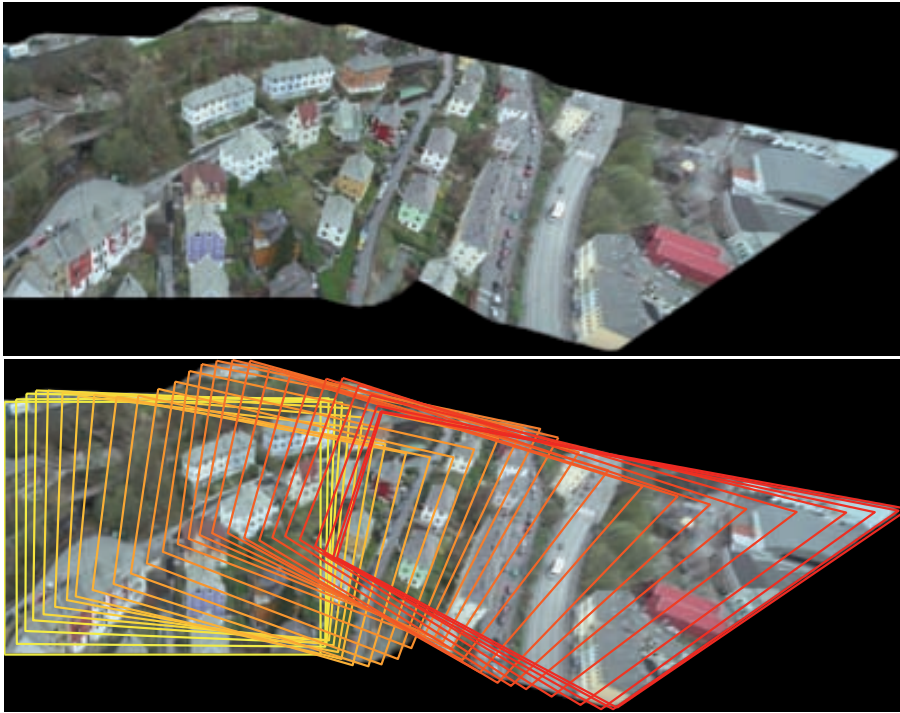


Figure 4.10: This example shows a 256-frame interlaced video, recorded from an UAV with a resolution of 720x576 pixels. A trapezoid is plotted every 10 frames showing the FOV, while the color of the trapezoid indicates time, where yellow marks the first frame and red the last one.

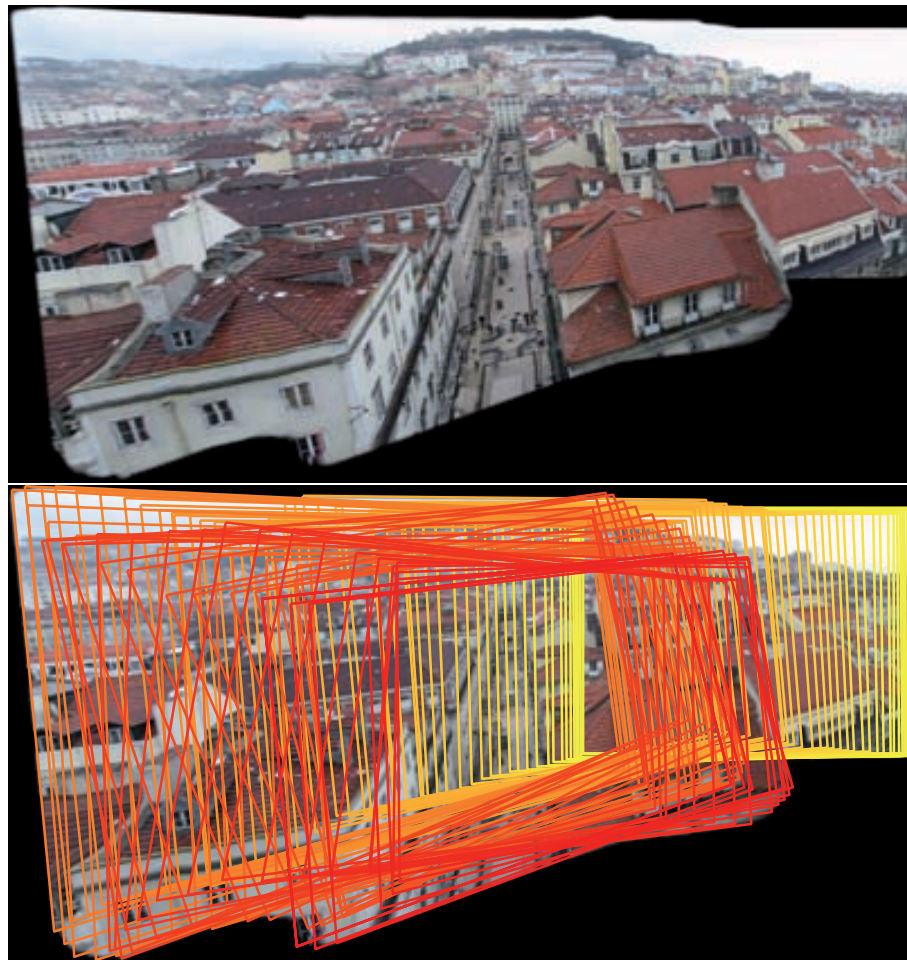


Figure 4.11: This example shows a 1000-frame progressive video, recorded using a hand-held photo camera (Cannon ixus 750) with a resolution of 640x480 pixels. A trapezoid is plotted every 10 frames showing the FOV, while the color of the trapezoid indicates time, where yellow marks the first frame and red the last one.

76DLIG: DIRECT LOCAL INDIRECT GLOBAL ALIGNMENT FOR VIDEO MOSAICING

Chapter 5

Real-Time Stereo Mosaicing using Feature Tracking

Until now, we have seen that by computing the 2D motion between frames it is possible to align the frames in a video sequence. However, the creation of multi-view stereo mosaics requires more accurate 3D motion computation. Fast and accurate computation of 3D motion is challenging in the case of unstabilized cameras moving in 3D scenes, which is always the case when stereo mosaics are used. Efficient blending of the mosaic strip is also essential.

Most cases of stereo mosaicing satisfy the assumption of limited camera motion, with no forward motion and no change in internal parameters. Under these assumptions, uniform sideways motion creates straight epipolar lines. When the 3D motion is computed correctly, images can be aligned in space-time volume to give straight epipolar lines, a method which is depth-invariant.

We propose to align the video sequence in a space-time volume based on efficient feature tracking, and in this chapter we used Kernel Tracking. Computation is fast as the motion is computed only for a few regions of the image, yet giving accurate 3D motion. This computation is faster and more accurate than the previously used direct approach.

We also present *Barcode Blending*, a new approach for using pyramid blending in video mosaics, which is very efficient. Barcode Blending overcomes the complexity of building pyramids for multiple narrow strips, combining all strips in a single blending step.

The entire stereo mosaicing process is highly efficient in computation and in memory, and can be run on mobile devices.

5.1 Introduction

In the previous chapters, we have presented techniques for mosaic construction for video sequences with negligible parallax effect. In such cases, only 2D motion is necessary to align the frames in a video sequence. However, when the camera has translational move-

ment, computing a 2D motion is not enough. In this chapter, we study the motion parallax caused by translational motion of the camera in order to create stereo mosaics using a single camera.

Stereo mosaics (or stereo panoramas) from a single camera are created by stitching together the video frames into at least two mosaic images that can create a 3D effect for the viewer. The 3D effect can be generated only when the captured scene has depth differences, and when the motion includes translation to create motion parallax.

Stereo mosaics from a single camera were first introduced in [64], [65], and were generated in two steps. The first step computed image translation and rotation between video frames, and the second step was mosaic stitching. Left eye mosaics were stitched from strips taken from the right side of the frames, and right-eye mosaics were generated from strips taken from the left side of the video frames. Generation of any number of views has been described in [68].

Computing a global image motion, e.g. a global rotation and translation between frames, results in distorted stereo mosaics, as relevant scenes have 3D parallax that can not be described by global parametric motion. The accuracy of motion computation can be increased when observing that the motion used for stereo mosaicing is limited: there is no forward motion, and there is no change in internal parameters. This observation was used in [71], [68] to propose depth invariant image alignment of the video sequence in an $x - y - t$ space-time volume, see Fig. 5.1.

The concept of Time Warping has been introduced based on the EPI (epipolar) planes described in [12]. Paper [12] observed that when the camera motion is constant, the lines generated by the trajectory of features in the EPI planes (epipolar lines) are straight lines (see Fig. 5.2a). In this case, by “slicing” the space-time volume we can generate a mosaic. However, in most of the cases the camera has variations in its motion, affecting the epilines’ trajectories and resulting in distorted mosaics. Time Warping was a resampling of the t axis in order to make the epipolar lines straight, simulating a constant velocity (see Fig. 5.2b). The method proposed in [71, 68] uses Lucas Kanade [52] over the entire image, making it computationally expensive.

As in [71, 68] we propose to align the video sequence using the $x - y - t$ space-time volume, but using efficient feature tracking instead of using Lucas Kanade. In chapter 4, Kernel Tracking has been demonstrated as a very efficient and accurate technique for mosaicing purposes. With our novel approach, only a few regions of each frame are needed in order to obtain the epipolar lines, resulting in real-time 3D alignment. All the alignment parameters can be computed in a single step using a Newton-style approximation, taking advantage of the character of the Kernel Tracking.

Another contribution in this chapter is the introduction of a new pyramid blending scheme

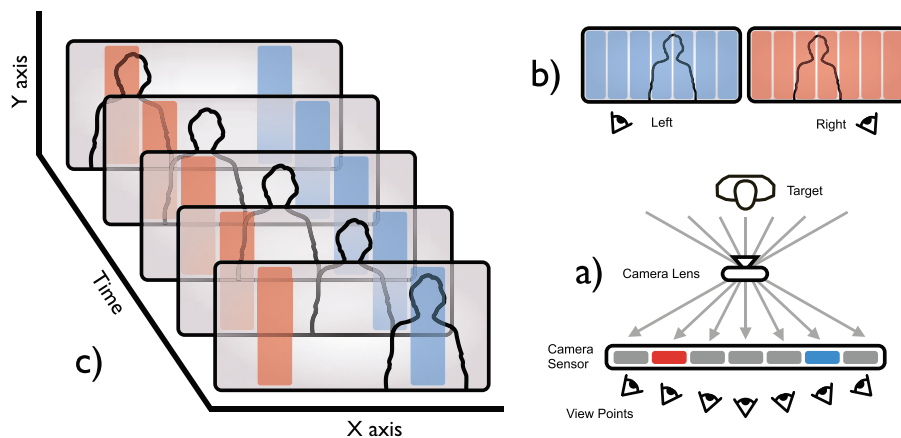


Figure 5.1: (a) Every region of the image sensor corresponds to a different viewing direction. (b) Stereo mosaics are generated by creating the left-eye mosaic from strips taken from the right side of the video frames, and the right-eye mosaic is created from strips taken from the left side of the video frames. (c) Stacking the video frames in an $x - y - t$ space-time volume. Creating a stereo mosaic can be seen as “slicing” the space-time volume by two $y - t$ planes. One on the left side and one on the right side.

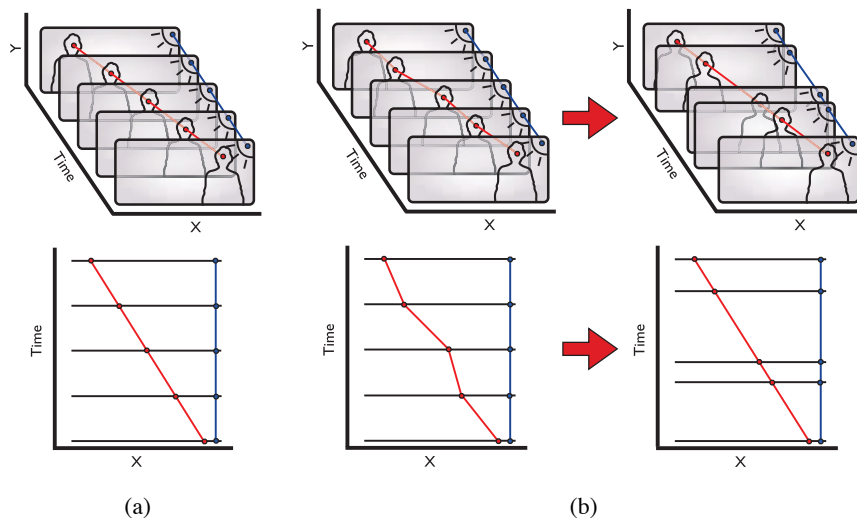


Figure 5.2: Resulting epilines when the camera has constant motion and when the camera has variations in its motion. In figure (a) an example of a sequence recorded camera that has constant motion is shown. While in figure (b) the same sequence is shown, but in this case the camera has variations in its motion. By resampling the t , Time Warping is able to straighten the epilines.

[14] for video mosaicing, overcoming the difficulty in blending together multiple narrow strips. The new blending, *Barcode Blending*, blends all strips in a single blending step.

The remainder of this chapter is structured as follows: Section 5.2 provides a detailed description of our image alignment approach, and section 5.3 presents mosaic generation and Barcode Blending. Finally, in section 5.4 the experiments conducted are described.

5.2 3D Motion Computation

In stereo mosaicing, when creating multiple views of a 3D scene, 2D image alignment is not appropriate. Computation of the 3D camera motion is needed for the creation of undistorted stereo mosaics. Following [71, 68], we align frames in the space-time volume so that EPI lines [12] become straight.

The alignment parameters of each frame f_i are $\Omega_i = \{t_i, y_i, \theta_i\}$, placing the frame f_i in the $x - y - t$ space-time volume such that the epipolar lines become straight. Parameters $\{t_i, y_i, \theta_i\}$ correspond to the time, the vertical displacement, and the rotation angle of frame f_i . We refer to this alignment as the Time Warping motion model.

In the next sections, we will first describe in detail our Time Warping approach. Afterwards, a summary of our algorithm will than be provided.

5.2.1 Time Warping Motion Model

Following [12], EPI lines in a video sequence are trajectories of feature points in (x, y, t) 3D space, where t is the time of each frame. When the camera undergoes translation in a constant velocity in a static scene, all EPI lines are straight lines (see Fig. 5.3). The slope of each line represents the depth of the corresponding feature point, where very far feature points will create lines almost perpendicular to the $x - y$ planes, while closer feature points will create a smaller angle with the $x - y$ planes. If we assume that the camera motion is a purely constant horizontal translation, the location of the feature point moves by a constant x displacement m between successive frames, where m corresponds to the depth of the point (the smaller m , the more the deep point). Formally, the EPI lines in the $x - y - t$ space-time volume can be recursively defined in a matrix form as:

$$\begin{pmatrix} x_i \\ y_i \\ t_i \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & m \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i-1} \\ y_{i-1} \\ t_{i-1} \\ 1 \end{pmatrix}, \quad (5.1)$$

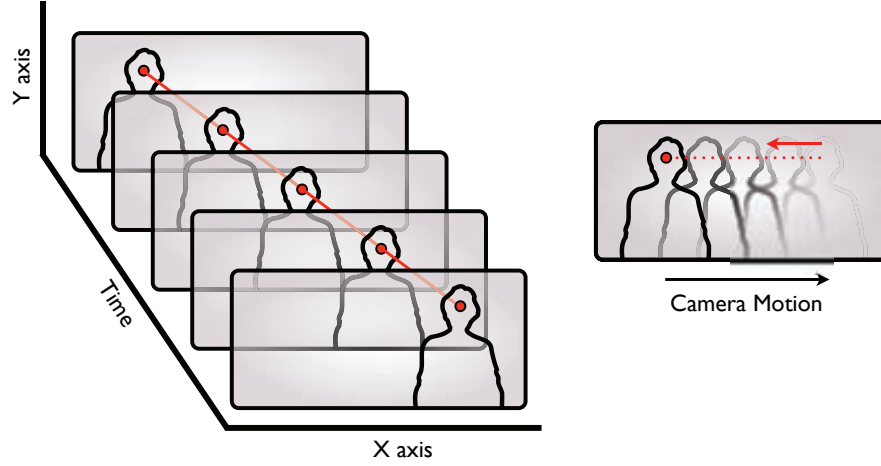


Figure 5.3: Scheme of how tracking a target can describe an EPI line in the $x - y - t$ space-time volume. The red line represents an EPI line, and the rectangles represents frames of a video sequence panning from left to right. The red circle represents the target.

where m is the slope of the EPI line in the (x, t) axis and i is the frame number. We use homogeneous coordinates since the transformation involves displacements. In this case, if we “slice” the space-time volume with a $y - t$ plane we get a reasonable mosaic image of the scene. But in most cases the motion of a hand-held camera is not constant and has rotations and vertical displacements. In such a cases, EPI lines can be defined recursively in a matrix form as:

$$\begin{pmatrix} x_i \\ y_i \\ t_i \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) & 0 & \frac{m}{\Delta t_i} \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) & 0 & \Delta y_i \\ 0 & 0 & 1 & \Delta t_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i-1} \\ y_{i-1} \\ t_{i-1} \\ 1 \end{pmatrix}, \quad (5.2)$$

where Δt_i , Δy_i and $\Delta\theta_i$ represent the motion parameters of frame f_i : the horizontal translation of the camera, the vertical image displacement, and the image rotation about the optical axis. These parameters are identical for all tracked feature points in image f_i , and represent the increments of these parameters with respect to the previous image. Our goal is to compute the parameters Δt_i , Δy_i and $\Delta\theta_i$ (three parameters for each image) such that the EPI lines become straight, and Eq. 5.2 is satisfied. This is equivalent to compensating for the vertical displacements and rotations of the image, and compensating for the horizontal translation of the camera by Time Warping.

Following the previous chapter of this thesis, we optimize the motion parameters using Kernel tracking with SSD, which was introduced in chapter 3. As a reminder, in Kernel Tracking the center c of the Kernel function was defined in terms of x and y coordinates and

was optimized to minimize the Matusita metric, which is defined as:

$$O(\mathbf{c}) = \|\sqrt{\mathbf{q}} - \sqrt{\mathbf{p}(\mathbf{c})}\|^2. \quad (5.3)$$

We would now like to optimize equation 5.3 in terms of t , y and θ . We use equation 5.2 to find the center of the kernel function \mathbf{c} in the image frame and the Jacobian of the Kernel Function, which is computed from the gradients of the Time Warping parameters as follows:

$$\mathbf{J}^{\mathbf{K}} = \left[\frac{\partial \mathbf{K}}{\partial t}, \frac{\partial \mathbf{K}}{\partial \mathbf{c}_y}, \frac{\partial \mathbf{K}}{\partial \theta} \right] = \left[\mathbf{J}_t^{\mathbf{K}}, \mathbf{J}_y^{\mathbf{K}}, \mathbf{J}_\theta^{\mathbf{K}} \right]. \quad (5.4)$$

where $\mathbf{J}_t^{\mathbf{K}}$ and $\mathbf{J}_\theta^{\mathbf{K}}$ can be computed using the chain rule as:

$$\frac{\partial \mathbf{K}}{\partial \lambda} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_{(x,y)}} \frac{\partial \mathbf{c}_{(x,y)}}{\partial t} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_x} m \quad (5.5)$$

$$\frac{\partial \mathbf{K}}{\partial \theta} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_{(x,y)}} \frac{\partial \mathbf{c}_{(x,y)}}{\partial \theta} = \frac{\partial \mathbf{K}}{\partial \mathbf{c}_x} (y - C_y) - \frac{\partial \mathbf{K}}{\partial \mathbf{c}_y} (x - C_x) \quad (5.6)$$

where (C_x, C_y) is the frame center. In this case the output of Kernel Tracking becomes $\Delta \mathbf{c} = (\Delta t, \Delta y, \Delta \theta)$.

5.2.2 3D Alignment by Time Warping

Our algorithm is based on two alternating steps. In the first step we use Kernel Tracking to track L target regions which are distributed over the frame as described in chapter 3. The paths of these tracked regions, $\{k_j\}_{j=1:L}$, are used to estimate straight EPI lines by computing the global m_j slope for each track. The global slopes m_j are computed using an iterative mean of the target velocity in the $x - y - t$ space-time volume, $(\frac{\Delta x_j}{\Delta t_j})$.

The second step (Sec. 5.2.1) consists of computing the Time Warping motion parameters using the estimated slopes of the EPI lines as computed in the previous step.

It is important to maintain a uniform distribution of tracked target regions during the alignment process in order to get accurate motion estimation. We do this by dividing the frame into sections, requiring that one target will be selected from each section. If a section contains two or more targets, only a single target is used. In our implementation we used the target region belonging to the longest track. In addition, new targets are selected for those sections having no targets.

Targets giving inaccurate trajectories are also removed. This includes targets placed on uniform regions, targets with a high SSD error after alignment, or targets that overlap the



Figure 5.4: Evolution of some target regions in a video sequence. Each circle represents a target and the color its state. From green to red the SSD error (green less error) is represented. The targets detected on flat regions are pink, targets out of bounds are black and new targets white.

image boundaries. Targets with a high SSD error or targets overlapping the image boundaries are trivial to detect. Targets placed on flat regions are detected using $\kappa_S(\mathbf{M}_T\mathbf{M})$ (see Eq. 5.7), where in [26] it is demonstrated that the Schatten norm of the matrix $\mathbf{M}_T\mathbf{M}$ is related to the accuracy of the estimation. When $\kappa_S(\mathbf{M}_T\mathbf{M})$ is above a certain threshold it means that the target region is placed on a flat region. Note that this process is performed in the first step of the proposed algorithm where only x and y parameters are estimated, therefore the matrix $\mathbf{M}_T\mathbf{M}$ is a 2×2 symmetric and positive definite. In such a case $\kappa_S(A)$ can be easily computed as:

$$\kappa_S(A) = \frac{(a_{11} + a_{22})^2}{a_{11}a_{22} - a_{12}a_{21}}. \quad (5.7)$$

A summary of the proposed algorithm is shown in algorithm 3 and an example of the evolution of the tracks in the first step is shown in Fig. 5.4.

Algorithm 3: Stereo Mosaic construction using Feature Tracking

Data:

Video Sequence $F = \{f_i\}_{i=1:N}$

Trackers $K = \{k_j\}_{j=1:L}$

Alignment Parameters $\Omega_0 = \{0, 0, 0\}$

Result:

Stereo Mosaic image M

```

1 begin
2   for  $i \leftarrow 1$  to  $N$  do
3     Step 1. Track  $K$ ;
4     Step 2. Compute Time Warping  $\Omega_i$ ;
5      $\forall k_j \in K$  update  $m_j$ ;
6      $\forall k_j \in K$  replace if is necessary;
7     Update the mosaic  $M$  using  $f_{i-1}$  and  $\Omega_{i-1}$ ;

```

5.3 Mosaic Construction

Once the video sequence is aligned, the mosaic image is generated by concatenating strips from different frames. Each strip $s_i \in f_i$ is generated by cutting a region surrounding the intersection $I_i = f_i \cap P_{y-t}$ between a frame f_i and a plane P_{y-t} in the $x-y-t$ space-time volume (see Fig. 5.7(a)). The width of strip s_i depends on the image alignment and is defined as:

$$s_i = \{x \in f_i | [I_i - \frac{d_i}{2}, I_i + \frac{d_{i+1}}{2}]\}, \quad (5.8)$$

where $d_i = \mu_m \Delta t_i$, and μ_m is the mean of all EPI lines slopes m_j (see Fig. 5.7(b)). Simple stitching of the strips produces visible misalignments caused by the parallax effect, and strip

blending is needed.

In this work we present *Barcode Blending*, a novel blending method which is based on Pyramid Blending, but only a single blending step is required for blending all strips. This process is much faster and gives better results when compared to the traditional approach of blending each strip separately.

First of all we will introduce the Multi-Band Blending and then we will describe, in detail the *Barcode Blending*.

5.3.1 Multi-Band Blending

Usually when two images are stitched, the edge between them is visible. The mean of the overlapped regions removes the edge but can blur the result. Another possible solution is to add a transition function to either boundary sides. This function, weighs the contribution of each image depending on the distance. Adding a transition function can improve the visual perception, but in many cases this results in blurry regions over the boundary. If the transition function is very steep then the edge is visible and if the transition function has little slope, then the image becomes blurry. In Fig. 5.5a shows the resultant image after stitching two images using a transition function with a high slope, while in Fig. 5.5b shows the result by using a function with lower slope. Note that in the first figure, the edge between them is clearly visible, while in the next image it is not. However, the image becomes more blurry (see the rocks in the lower part of the image). Therefore, it is mandatory to select an appropriate slope in order to maximize the transition. However, this is practically impossible, because the image is formed by different frequencies and while high frequencies require high slopes, low frequencies require low slopes.

Multi-Band Blending [14], proposes to decompose the images into different frequency bands, in order to apply the transition function that better suits for each case. Once we have stitched each frequency band together the image is recomposed. To decompose the image into different bands, a set of low-pass filters are applied to generate a sequence of images in which the band limit is reduced from image to image in one-octave steps. Then the different frequency bands can be obtained by simply subtracting each low-pass image from the previous image in the sequence.

The low-pass filtered images G_0, G_1, \dots, G_N can be obtained by repeatedly convolving a Gaussian function and down sampling the image in order to get a new image with a half of the size. Burt et al.[14] defines this procedure as:

$$G_i = REDUCE[G_{i-1}]. \quad (5.9)$$

The resultant set of low-pass filters is called a Gaussian Pyramid, because the size of each

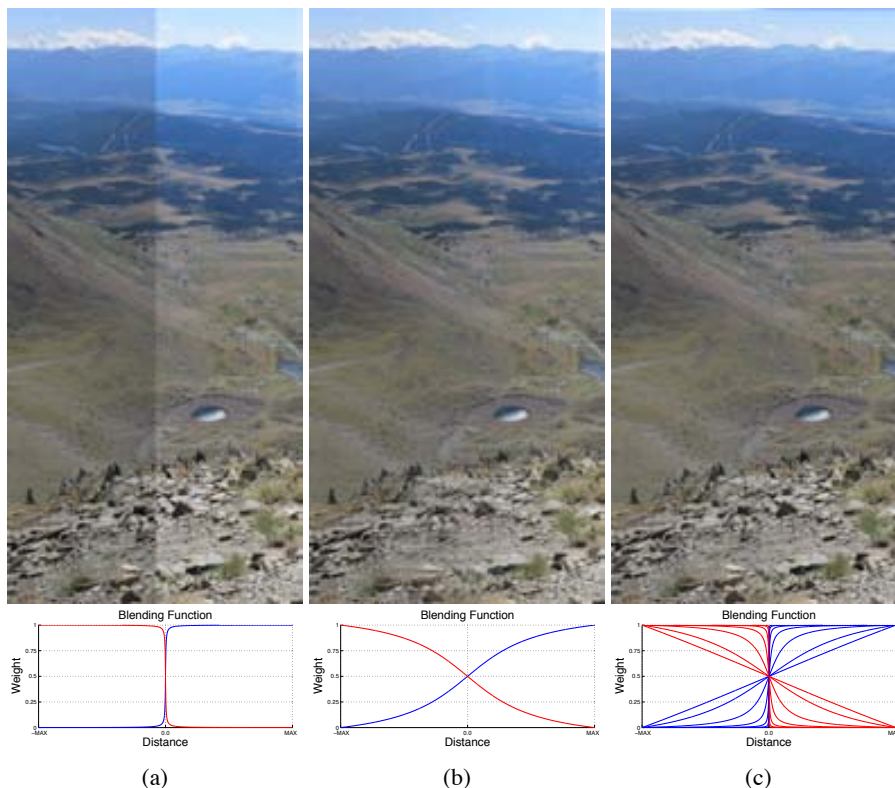


Figure 5.5: Comparison between three different blending functions. The top images show the results after applying the transition functions that are shown in the bottom figures, where the red line is the function applied to the left image and the blue line is the function applied to the right image. (a) shows the results of applying a transition function with high slope. As can be seen, the edge between the two images is visible. (b) is the result of stitching the same images using a transition function with lower slope. Note that in this case, the edge between the two images is no longer visible. However, the image becomes more blurry (see the rocks in the lower part of the image). Finally, in figure (c) shows the result after applying Multi-Band Blending. In this case it is not possible to detect that this image is formed from two different images. The blending function, in this case, is composed of multiple transition functions, one for each pyramid level. High image frequencies are stitched together using high slopes, while lower image frequencies are stitched together using lower slopes.

low-pass filter decreases at each iteration, forming an inverted pyramid. Note that, G_0 is the original image.

On the other hand, the different image frequency bands L_i are generated by subtracting two consecutive low-pass filters G_i and G_{i-1} . Note that G_{i-1} has half of the size of G_i and this makes it necessary to interpolate new pixels.

This process is defined as:

$$G'_{i-1} = EXPAND[G_{i-1}], \quad (5.10)$$

and is performed by copying the values of G_{i-1} to even positions of a matrix of twice its size. Then the pixel values of the rest of the positions are interpolated by convolving the image by the same Gaussian function used in the *REDUCE* operation. The resulting equation to compute the frequency bands is:

$$L_i = G_i - EXPAND[G_{i-1}], \quad (5.11)$$

and the set of L_0, L_1, \dots, L_N is called the Laplacian Pyramid.

Finally, the image is recomposed by adding all the Laplacian, by iteratively applying the *EXPAND* function and adding the result to the superior level in the pyramid. Fig. 5.6, shows a scheme of the different elements of this method. The left column can be seen the Gaussian pyramid and the right column shows the Laplacian pyramid. In addition, in Fig. 5.5c shows the result of applying Multi-Band Blending on the example presented in the first part of this section.

5.3.2 Barcode Blending

Since Multi-Band Blending is great to stitch together two images with high overlapping regions, it becomes complex and inefficient when it is applied directly to stitch multiple and small images. Multi-Band Blending requires big overlapping regions, because at each pyramid level the image size is reduced. In addition, the need for generating a pyramid level for each aligned frame increases the cost significantly. Furthermore, applying Multi-Band Blending on stereo mosaicing is especially difficult, because mosaics are generated by multiple and very small strips.

Barcode Blending is designed to simplify the mosaic construction using Multi-Band Blending and at the same time reduce the computational requirement. It starts with the generation of two different mosaics without blending, one from the odd frames and another for the even frames. Strips in the odd mosaic are defined as:

$$s_{2k+1}^{Odd} = \{x \in f_{2k+1} | [I_{2k+1} - d_{2k+1}, I_{2k+1} + d_{2k+2}]\}, \quad (5.12)$$

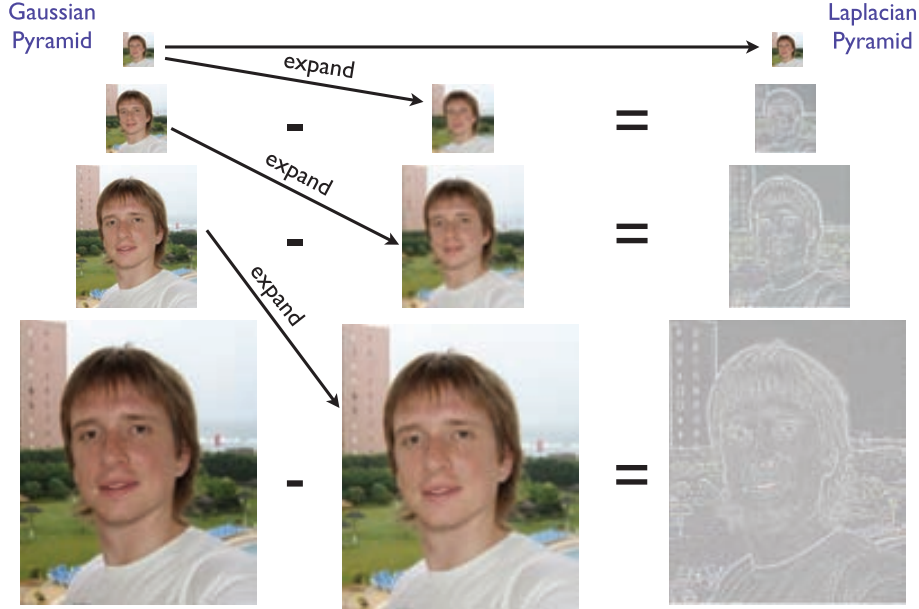


Figure 5.6: Example of Gaussian Pyramid and Laplacian pyramid. The first column of images (Gaussian pyramid) are subtracted to the second column (Expanded Gaussian pyramid) giving as a result the third column (the Laplacian pyramid).

and strips in the even mosaic are defined as:

$$s_{2k}^{Even} = \{x \in f_{2k}[[I_{2k} - d_{2k}, I_{2k} + d_{2k+1}]]\}, \quad (5.13)$$

where $k \in [0, N - 1]$. The number of strips in each of these two mosaics is half of the number of strips in the final mosaic image, and each strip has double the width. The final mosaic will be generated by pyramid blending from these two mosaics using a binary mask which defines which part of the final mosaic is taken from the odd mosaic and which is taken from the even mosaic. The mask therefore looks like a barcode whose strips have the same widths as the strips in the final mosaic, and is defined as,

$$s_i^{mask} = \begin{cases} [1, \frac{d_i}{2}] & = \begin{cases} 0 & \Leftarrow i \text{ Odd} \\ 1 & \Leftarrow i \text{ Even} \end{cases} \\ [\frac{d_i}{2}, d_i] & = \begin{cases} 1 & \Leftarrow i \text{ Odd} \\ 0 & \Leftarrow i \text{ Even} \end{cases} \end{cases}, \quad (5.14)$$

Fig. 5.7(c) shows the Barcode Blending process.

Once we have generated the odd mosaic, the even mosaic, and the mask, we blend the two mosaics according to the mask using a simple pyramid blending. Some results are shown in Fig. 5.8 and Fig. 5.9. It is important to note that generating the two mosaics and the mask is a trivial and fast process, and that pyramid blending is used only once.

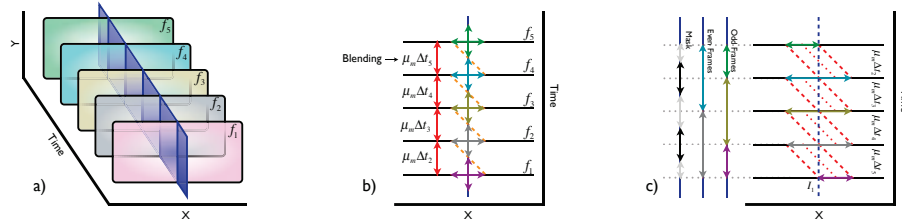


Figure 5.7: (a) Mosaic construction as slicing the $x - y - t$ space-time volume. Five frames of a video sequence are shown in an aligned space-time volume. The generated mosaic is the blue plane P_{y-t} . Each frame is represented by a different color. (b) Image strips around the intersection of P , with each image are pasted onto P . In order to simplify the scheme, we assume that the slope of the epipolar lines $\mu_m = 1$. (c) Odd, even and mask mosaics generation, in order to compute Barcode Blending.

5.4 Experimental Results

In all the examples we used a uniform lattice of 12×12 tracked features. The use of anaglyph glasses will enable the reader to appreciate the 3D effect.

Fig. 5.10 used a video captured from a river boat. A small area is shown from different mosaics to emphasize the parallax effect between the different mosaic images.

The mosaics in Fig. 5.11 were made from video recorded by a hand held iPhone 4. The resolution of the captured sequences is 1280×720 .

We have implemented the proposed method in C++, and tested it by tracking 144 features (on a 12×12 lattice) on three HD video sequences (1280×720) running on a Mac Book Pro 15' 2.66GHz Core i7. The alignment frame rate was ≈ 18 fps, with implementation not fully optimized, no hardware acceleration, and using a single thread. The most costly step was the tracking of 144 features, each of size 31×31 pixels, running at ≈ 30 fps. The 3D alignment is computed at a frame rate of ≈ 45 fps.

Furthermore, we have developed an iPhone/iPad and Android¹ application called Snapily3D; a screen shot of the application, can be seen in Fig. 5.12. The application can be downloaded here <http://itunes.apple.com/us/app/snapily3d/id473670116?mt=8>.

¹It is under development.

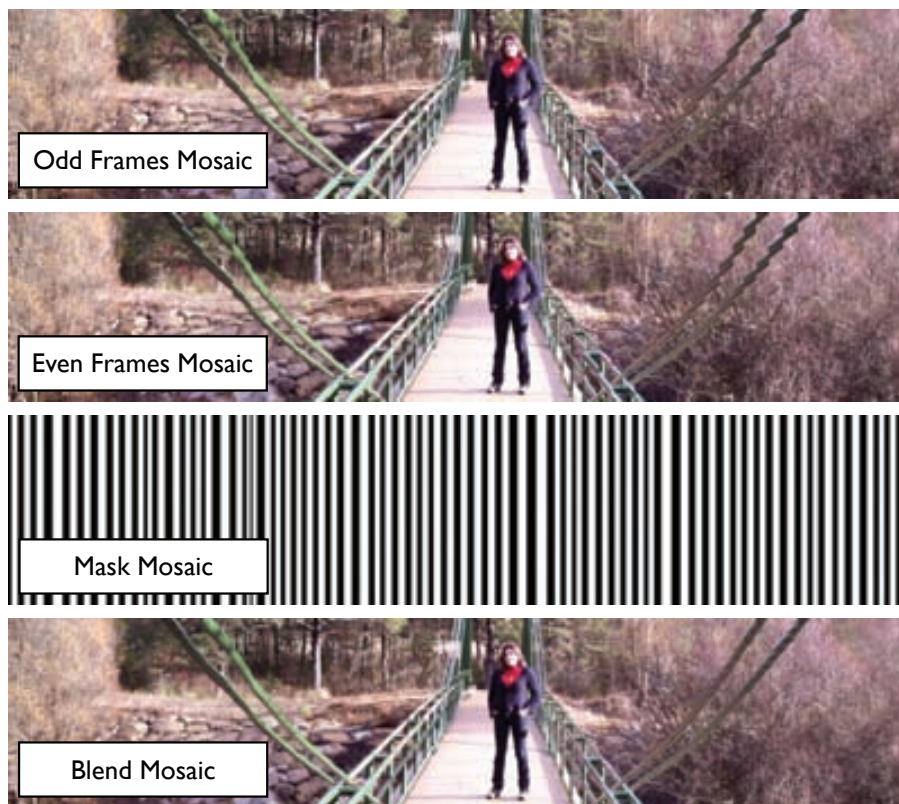


Figure 5.8: Example of odd, even and mask mosaic and the resultant mosaic when Barcode Blending is applied.



Figure 5.9: Comparison between mosaic regions with and without Barcode Blending. The top images belong to a mosaic image generated without any blending and the bottom images are generated using Barcode Blending.



Figure 5.10: An anaglyph stereo mosaic generated from a video sequence of 450 frames captured from a river boat. The bottom images show the same regions as they appear in the different mosaics, showing different points of view generated using the same aligned volume.



Figure 5.11: Anaglyph stereo mosaics generated from videos recorded by a hand held iPhone 4, with resolution of 1280×720 pixels.



Figure 5.12: Snapily3D iPhone application developed using the technology presented in this chapter.

Chapter 6

Conclusions and Future Work

In this chapter we summarize the main contributions of each chapter with a discussion of the key features of our work. In addition, we present a list of open lines of research for future work.

This chapter is structured as follows, the conclusions of each chapter are presented in different sections, starting by chapter 2. Finally at the end of this chapter is presented the lines of research for a future work.

6.1 Background Subtraction for Static Cameras based on MRF

This thesis starts by solving the problem of motion detection through background subtraction. The majority of background subtraction algorithms present in the state of the art have been described and we have seen how these methods can be divided into two main groups: pixel-level and frame-level. The main difference of these two groups is the use of spatial information (frame-level uses it and pixel-level does not). The results show that the spatial information improves the final segmentation; however, the computational cost is increased substantially. For these reasons the two groups are perfectly valid and the use of one approach or other depends on the problem.

The work presented in this chapter, is focussed on extending a frame-level background subtraction method based on Markov Random Fields (MKF). Markov networks, configured as a uniform lattice connecting adjacent pixels, perfectly suits on the problem of background subtraction since it can be used to directly add spatial information to pixel-level methods. The results presented show that, in all the pixel-level methods studied, MKF improves the final mask. The main problem is the important increase of computational cost. The optimal

state of each node in the network was computed using an approximation of the belief propagation algorithm, which could be the main cause of the growth of the computational cost. The possibility of using other methods such as graph cuts could be a possible future study.

The main contribution in this chapter is to provide a framework for combining different sources of information using an adapted version of the MKR formulation. To this end, we generate a single discrete function converts the inputs of different binary detectors to a probability of being background or foreground. Using a single discrete function instead of multiple functions for each binary detector has a significant impact into the formulation, since it becomes general and scalable for more complex models. If the only presence of a MKF improves the final segmentation, the use of multiples sources of information gives even better results. The main challenge of our approach is to select the correct information sources, which are complementary for improving the final motion mask. And again, the computational cost is increased, since it is necessary to compute the output of each detector.

6.2 Background Subtraction for Non-static cameras based on MKT

The main limitation of the method presented in chapter 2, and in general for all background subtraction methods, is the lack of robustness when the camera is not static. For instance, when the video sequence is recorded using PTZ cameras. In such a case, it is necessary to compensate for the motion induced by the camera movement, before applying background subtraction. Otherwise, the background image becomes blurry.

We have shown that it is possible to compensate the camera motion by tracking the whole frame, when the major part of it belongs to the background. The method proposed is based on Multiple Kernel Tracking, which lets us compute the 2D camera motion with high efficiency and accuracy. In this chapter, we show how to optimize two different motion models: a translational model, which can compute displacements on the X and Y axis; and an affine motion model, which adds rotation and escalations to the previous motion model. Multiple Kernel tracking has low memory requirements, since the target regions are represented by histograms (in our case, histograms of 12 bins). In addition, the computation is extremely fast and converges to a solution in a few iterations (from 5 to 10). This fact is remarkable since other methods, like Lucas Kanade tracking, need to store all the pixel values of the target region and require an average of 60 iterations to converge.

We have also shown that it is not sufficient to use the coordinates of a single frame to compute the background image on PTZ-like cameras. Since motion detection can only be applied to those regions present in the background image (as in this case) the major part of the scene is not represented. To overcome this problem, we use the background mosaic, which is the result of incrementally adding to the background image the regions without representation.

The main drawback of the method proposed is the 2D motion model used (translational and affine). Aligning frames using these motion models produces misalignments, since a flat plane tilted with respect to the camera requires projective transformations.

6.3 DLIG: Direct Local Indirect Global alignment for Video Mosaicing

The main aim in chapter 4 was to increase the accuracy when constructing the mosaic image. To this end, we have presented DLIG alignment, a new alignment method which combines the precision of direct methods with the simplicity of estimating complex transformations of indirect methods.

In addition, we have shown how to deal with the two main problems that arise when constructing the mosaic image: the error accumulation and the precision degeneration. The first is caused by accumulating error when computing the 2D motion over the video sequence. Even having an alignment method with high accuracy, a small error is produced, which accumulates with the other errors produced by each frame in the video sequence produces huge misalignment, especially when the camera has circular paths. We have shown how to minimize this problem by computing the alignment using regions from multiple frames (not only the previous one).

On the other hand, the precision degeneration is caused by aligning warped frames, where the error produced when aligning patches on different parts of the frame depends on the amount of deformation produced by the warp. This problem is typically produced when the mosaic image is used to align the frames. We have shown that keeping an unwrapped representation of the the mosaic let us to reduce the precision degeneration due to the frame locality property.

Additionally, we take benefit from the frame locality property to embed a tracking algorithm, to compute the direct local part of the DLIG alignment. We have also shown that it is possible to deal with incorrect matching by using weighted alignment estimation, and how to deal with background changes using a model update strategy.

We have tested different tracking algorithms, concluding that Kernel Tracking is the most suitable. It can deal with lighting changes in progressive and interlaced videos with high accuracy and low computational cost and memory requirements.

The main drawback of our approach is that it does not work with low resolution images, as fewer regions used to track the less accurate the alignment becomes. DLIG alignment uses

regions which are considerably large compared to other methods. However, nowadays most of the cameras are HD. The minimum working resolution for the DLIG alignment is 640x480 (VGA), but it is recommended to use a resolution of at least 800x600.

6.4 Real-Time Stereo Mosaicing using Feature Tracking

We end this thesis by exploring the case of generating mosaics, when the camera has translational movement. In such case, presence of parallax let us create multiple views of the same scene. Then a stereo mosaic can be generated by selecting two views corresponding to the right and left eyes.

The main contribution of chapter 5 is to estimate epipolar lines by tracking different regions over the video sequence into the space-time volume representation. Once we have multiple epipolar lines we estimate a transformation to straighten its trajectories.

The presented stereo mosaicing method is faster and more accurate than previous approaches. In our case, the 3D alignment is based on feature tracking, which allows us to process HD video sequences with low computational cost and memory requirements. In this chapter we proposed to use Kernel Tracking, but any other feature tracking can be used.

In addition we have presented a new blending method, the Barcode Blending, that perfectly fits the problem of blending multiple frame strips. Barcode Blending is based on multi-band blending but presents a substantial improvement in speed and memory when used for mosaic blending, since only one Laplace of Gaussian Pyramid is needed. Furthermore, it is trivial to implement and results are better than traditional mosaic blending.

Finally, we have shown that this algorithm can be used in mobile phones, through to the implementation of an iPhone and an Android applications.

6.5 Future Work

As a future study, we propose to implement the presented algorithm using CUDA, OpenCL or OpenGL shaders (GLSL). Due to the nature of the GPU it is possible to process a huge number of task in parallel. This property perfectly suits the algorithms presented in this work, since they are highly parallelizable. For instance, in the first chapter the process can be parallelized at pixel-level and for the other chapters the regions can be also tracked also in parallel. Initial tests confirms that a GPU implementation can be 100 times faster.

Another future line of study is to increase the accuracy of Kernel Tracking by combining its result with a Lucas Kanade approach. When Lucas Kanade has a close initialization to its final position, it becomes much more accurate than Kernel Tracking.

Dealing with flat regions is also problematic in the proposed mosaicing methods, we tried Kalman filters to deal with this, but was not always effective. It should be interesting to work on this problem in the future.

Mosaicing can be used for many purposes and in this thesis we only deal with some of them. It could be very interesting to use the frames aligned into a space-time volume to summarize surveillance video sequences. There is some published work dealing with this problem using static cameras, but none yet for PTZ cameras.

Publications

Publications as first author:

- **M. Vivet**, S. Peleg, X. Binefa. Real-Time Stereo Mosaicing using Feature Tracking, In *IEEE International Workshop on Video Panorama*, pp. 577-582, 2011.
- **M. Vivet**, B. Martinez, X. Binefa. DLIG: Direct Local Indirect Global Mosaicing using a Region Tracer based model. In *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1869-1878, 2011.
- **M. Vivet**, B. Martinez, X. Binefa. Real-Time Motion Detection for a Mobile Observer Using Multiple Kernel Tracking and Belief Propagation. In *Iberian Conference on Pattern Recognition and Image Analysis (IbPria)*, pp. 144-151, 2009.
- **M. Vivet**, B. Martinez, X. Binefa. Multiple Cue Data Fusion using Markov Random Fields for Motion Detection. In *International Conference on Computer Vision Theory and Applications*, pp. 529-534, vol. 2, 2009.

Publications as second author:

- B. Martinez, **M. Vivet**, X. Binefa. Compatible Particles for Part-based Tracking. In *Articulated Motion and Deformable Objects, Conference on (AMDO)*, pp. 1-10, 2010.

Bibliography

- [1] A Aner and J R Kender. Mosaic-based clustering of scene locations in videos. *IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 111–117, 2001.
- [2] P Azzari, L Di Stefano, and A Bevilacqua. An effective real-time mosaicing algorithm apt to detect motion through background subtraction using a PTZ camera. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 511–516, 2005.
- [3] S Baker and I Matthews. Equivalence and efficiency of image alignment algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1090–1097, February 2001.
- [4] S Baker and I Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56:221–255, 2004.
- [5] P Bao and D Xu. Panoramic image mosaics via complex wavelet pyramid. In *IEEE International Conference on System, Man, and Cybernetics*, pages 4614–4619, 1998.
- [6] H Bay, T Tuytelaars, and L Gool. SURF: Speeded Up Robust Features. In *Computer Vision–ECCV 2006*, pages 404–417. European Conference on Computer Vision, Berlin, Heidelberg, 2006.
- [7] A Bevilacqua and P Azzari. High-Quality Real Time Motion Detection Using PTZ Cameras. In *IEEE International Conference on Video and Signal Based Surveillance*, page 23, 2006.
- [8] A Bevilacqua and P Azzari. A fast and reliable image mosaicing technique with application to wide area motion detection. In *International Conference Image Analysis and Reconstruction*, pages 501–512, 2007.
- [9] K S Bhat, M Saptharishi, and P K Khosla. Motion detection and segmentation using image mosaics. In *IEEE International Conference on Multimedia and Expo*, pages 1577–1580, 2000.
- [10] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, London, 2006.
- [11] A Blake, P Kohli, and Carsten Rother. *Markov Random Fields for Vision and Image Processing*. MIT Press (MA), July 2011.

- [12] R Bolles, H Baker, and D Marimont. Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion. *International journal of computer vision*, pages 7–55, 1987.
- [13] M Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.
- [14] P Burt and E Adelson. A Multiresolution Spline with Application to Image Mosaics. *Acm Transactions on Graphics*, 2(4):217–236, 1983.
- [15] D Butler and S Sridharan. Real-Time Adaptive Background Segmentation. *Proceedings of the International Conference on Multimedia and Expo*, 3:341–344, 2003.
- [16] F Caballero, L Merino, J Ferruz, and A Ollero. Homography Based Kalman Filter for Mosaic Building. Applications to UAV position estimation. In *IEEE International Conference on Robotics and Automation*, pages 2004–2009, 2007.
- [17] D Chen, S Denman, C Fookes, and S Sridharan. Accurate Silhouettes for Surveillance - Improved Motion Segmentation Using Graph Cuts. In *International Conference on Digital Image Computing: Techniques and Applications*, 2010.
- [18] R. T. Collins, A. J. Lipton, T Kanade, and etal. A System for Video Surveillance and Monitoring. Technical Report CMU-RI-TR-00-12, The Robotics Institute, Pittsburgh, May 2000.
- [19] D Comaniciu, V Ramesh, and P Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 142–149, 2000.
- [20] C Correa and K Ma. Dynamic video narratives. *Acm Transactions on Graphics*, 29, 2010.
- [21] R Cucchiara, C Grana, G Neri, M Piccardi, and A Prati. The Sakbot system for moving object detection and tracking. In *Proceedings European Workshop on Advanced Video-based Surveillance*, 2001.
- [22] R Cucchiara, C Grana, M Piccardi, and A Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 25(10):1337–1342, 2003.
- [23] F Dellaert and R Collins. Fast image-based tracking by selective pixel integration. In *Proceedings of the International Conference on Computer Vision*, 1999.
- [24] S.M Desa and Q.A Salih. Image subtraction for real time moving object extraction. In *Proceedings of the International Conference on Computer Graphics, Imaging and Visualization*, pages 41–45, 2004.
- [25] A Elgammal and D Harwood. Non-parametric Model for Background Subtraction . In *European Conference in Computer Vision*, pages 751–767, 2000.

- [26] Z Fan, M Yang, Y Wu, G Hua, and T Yu. Efficient Optimal Kernel Placement for Reliable Visual Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–665, 2006.
- [27] P.F Felzenszwalb and D.R Huttenlocher. Efficient belief propagation for early vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.
- [28] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.
- [29] A Gademer, F Mainfroy, L Beaudoin, and al, et. Faucon noir UAV project development of a set of tools for managing, visualizing and mosaicing centimetric UAV images. *IEEE International Geoscience and Remote Sensing Symposium*, 3, 2009.
- [30] R N Gracias and J Santos-Victor. Underwater Video Mosaics as Visual Navigation Maps . *Computer Vision and Image Understanding*, 79:66–91, 2000.
- [31] C Guillot, M Taron, P Sayd, and al, et. Background subtraction adapted to PTZ cameras by keypoint density estimation. In *Proceedings of the British Machine Vision Conference*, pages 34.1–34.10. British Machine Vision Association, 2010.
- [32] G D Hager and P N Belhumeur. Efficient region tracking with parametric models of geometry and illumination . *IEEE Transactions on Pattern Analysis Machine Intelligence*, 1998.
- [33] G D Hager, M Dewan, and C Stewart. Multiple kernel tracking with SSD. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 790–797, 2004.
- [34] B Han and D Comaniciu. Sequential kernel density approximation through mode propagation: applications to background modeling. *Asian Conference on Computer Vision*, 2004.
- [35] I Haritaoglu and D Harwood. Active Outdoor Surveillance . *International Conference on Image Analysis and Processing*, pages 1096–1099, 1999.
- [36] E Hayman and J O Eklundh. Statistical Background Subtraction for a Mobile Observer . *IEEE International Conference on Computer Vision*, 1:67–74, October 2003.
- [37] L Heng-hui, Y Jin-Feng, R Xiao-hui, and W Ren-biao. An improved mixture-of-Gaussians model for background subtraction. In *International Conference on Signal Processing*, 2007.
- [38] J Hong, W Lin, H Zhang, and L Li. Image Mosaic Based on SURF Feature Matching. In *International Conference on Information Science and Engineering*, pages 1287–1290, December 2009.
- [39] M Irani and P Anandan. Video indexing based on mosaic representations. *Proceedings of the IEEE*, 86:905–921, 1998.
- [40] M Irani, S Hsu, and P Anandan. Video compression using mosaic representations. *Signal Processing: Image Communication*, 7:529–552, 1995.

- [41] E Kang, I Cohen, and G Medioni. A graph-based global registration for 2D mosaics . In *International Conference on Pattern Recognition*, pages 1257–1260, 2000.
- [42] J Kato, T Watanabe, and S Joga. An HMM-Based Segmentation Method for Traffic Monitoring Movies . *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.
- [43] R Kindermann and J Laurie Snell. *Markov random fields and their applications*. American Mathematical Society, 1rst edition, 1980.
- [44] P Kohli and P.H.S Torr. Efficiently solving dynamic Markov random fields using graph cuts. In *IEEE International Conference on Computer Vision*, pages 922–929, 2005.
- [45] D Koller, J Weber, T Huang, J Malik, and etal. Toards robust automatic traffic scene analysis in real-time . In *International Conference on Pattern Recognition*, pages 126–131, 1994.
- [46] D S Lee, J J Hull, and B Erol. A Bayesian Framework For Gaussian Mixture Background Modeling. *IEEE International Conference on Image Processing*, 2003.
- [47] M Lee, W Chen, C Lin, C Gu, and etal. A layered video object coding system using sprite and affine motion model. *IEEE Transactions on Circuits and Systems for Video Technology*, 7:130–145, 1997.
- [48] J. P. Lewis. Fast Normalized Cross-Correlation. *Vision Interface*, pages 120–123, 1995.
- [49] R Li, Y Chen, and Z Xudong. Fast Robust Eigen-Background Updating for Foreground Detection. In *IEEE International Conference on Image Processing*, 2006.
- [50] B.P.L Lo and S.A Velastin. Automatic congestion detection system for underground platforms. In *Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing*, pages 158–161, 2001.
- [51] S Lovegrove and A J Davison. Real-time spherical mosaicing using whole image alignment. In *European Conference on Computer Vision*, pages 73–86. Springer-Verlag, September 2010.
- [52] B D Lucas and T Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conferences on Artificial Intelligence*, pages 674–679, 1981.
- [53] David J C MacKay. *Information theory, inference, and learning algorithms*. Cambridge, October 2003.
- [54] R Marzotto, A Fusiello, and V Murino. High resolution video mosaicing with global alignment. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2004.
- [55] L Matthews, T Ishikawa, and S Baker. The Template Update Problem . *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:810–815, 2004.

- [56] N J B McFarlane and C P Schofield. Segmentation and tracking of piglets in images. *Machine Vision and Applications*, 8(3):187–193, May 1995.
- [57] G Medioni, I Cohen, F Bremond, S Hongeng, and R Nevatia. Event detection and analysis from video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):873–889, 2001.
- [58] A Mittal and D Huttenlocher. Scene modeling for wide area surveillance and image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 160–167, 2000.
- [59] B Morse, C Engh, and M Goodrich. UAV video coverage quality maps and prioritized indexing for wilderness search and rescue. *IEEE International Conference on Human-Robot Interaction*, pages 227–234, 2010.
- [60] A Nemra and N Aouf. Robust invariant automatic image mosaicing and super resolution for UAV mapping. *International Symposium on Mechatronics and its Applications*, 2009.
- [61] A Neri, S Colonnese, G Russo, and P Talone. Automatic moving object and background separation. *Signal Processing*, pages 219–232, 1998.
- [62] E Nunes, A Conci, and A Sanchez. Robust background subtraction on traffic videos. In *International Conference on Systems, Signals and Image Processing*, 2010.
- [63] N.M Oliver, B Rosario, and A.P Pentland. A Bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 22(8):831–843, 2000.
- [64] S Peleg and M Ben-Ezra. Stereo panorama with a single camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1395–1401, 1999.
- [65] S Peleg, M Ben-Ezra, and Y Pritch. Omnistereo: panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):279–290, March 2001.
- [66] Y Pritch, A Rav-Acha, and A Gutman. Webcam Synopsis: Peeking Around the World . *IEEE International Conference on Computer Vision*, 2007.
- [67] R.J Qian and T.S Huang. Object detection using hierarchical MRF and MAP estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 186–192, 1997.
- [68] A Rav-Acha and S Peleg. A unified approach for motion analysis and view synthesis . In *International Symposium on 3D Data Processing, Visualization and Transmission*, pages 717–724. IEEE, 2004.
- [69] A Rav-Acha, Y Pritch, D Lischinski, and S Peleg. Dynamosaicing: Mosaicing of dynamic scenes. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 29:1789–1801, 2007.

- [70] A Rav-Acha, Y Pritch, and S Peleg. Making a Long Video Short: Dynamic Video Synopsis . *IEEE International Conference on Computer Vision and Pattern Recognition*, 2006.
- [71] A Rav-Acha, Y Shor, and S Peleg. Mosaicing with parallax using time warping. In *IEEE Conference on Computer Vision and Pattern Recognition Workshop*, 2004.
- [72] J Rittsher, J Kato, S Joga, and A Blake. A Probabilistic Background Model for Tracking. In *European Conference in Computer Vision*, pages 336–350, 2000.
- [73] J Rymel, J Renno, D Greenhill, J Orwell, and G A Jones. Adaptive eigen-backgrounds for object detection. In *IEEE International Conference on Image Processing*, 2004.
- [74] H Sawhney. True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(0162-8828):235–243, 1999.
- [75] H Sawhney, S Hsu, and R Kumar. Robust video mosaicing through topology inference and local to global alignment. In *European Conference on Computer Vision*, pages 103–119, 1998.
- [76] H Y Shum and R Szeliski. Construction of panoramic image mosaics with global and local alignment . *International journal of computer vision*, 2001.
- [77] A Smolic and T Wiegand. High-resolution video mosaicing. In *International Conference on Image Processing*, 2001.
- [78] C Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 2246, 1999.
- [79] Z Tao, F Shumin, L Xiaodong, and J Hong. Moving target detection in image sequences. In *Chinese Control Conference*, 2008.
- [80] C Tomasi and J Shi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.
- [81] P.H.S Torr and A Zisserman. MLESAC: A New Robust Estimator with Application to Estimating Image Geometry . *Computer Vision and Image Understanding*, 78(1):138–156, April 2000.
- [82] K Toyama, J Krumm, B Brumitt, and B Meyers. Wallflower: principles and practice of background maintenance. In *International Conference on Computer Vision*, pages 255–261, 1999.
- [83] M Unger, M Asbach, and P Hosten. Enhanced background subtraction using global motion compensation and mosaicing. In *IEEE International Conference on Image Processing*, pages 2708–2711, 2008.
- [84] D Wang, T Feng, H Shum, and S Ma. A Novel Probability Model for Background Maintenance and Subtraction . In *International Conference on Video Interface*, 2002.

- [85] W Wei, H Jun, and T Yiping. Image Matching for Geomorphic Measurement Based on SIFT and RANSAC Methods. In *Proceedings of the International Conference on Computer Science and Software Engineering*, pages 317–332, 2008.
- [86] Yair Weiss and William T Freeman. Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology. *Neural Computation*, 2001.
- [87] A P Whichello and H Yan. Document image mosaicing. *International Conference on Pattern Recognition*, 2:1081–1083, August 1998.
- [88] F Winkelman and I Patras. Online globally consistent mosaicing using an efficient representation. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3116–3121, October 2005.
- [89] C Wren, A Azarbayejani, T Darrell, and A Pentland. Pfinder: real-time tracking of the human body. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, pages 51–56, 1996.
- [90] C R Wren, A Azarbayejani, T Darrell, and A.P Pentland. Pfinder: real-time tracking of the human body. *IEEE International Conference on Pattern Analysis and Machine Intelligence*, 19(7), 1997.
- [91] J.S Yedidia, W.T Freeman, and Y Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.
- [92] Z Yi and F Liangzhong. Moving object detection based on running average background and temporal difference. In *International Conference on Fan Liangzhong Intelligent Systems and Knowledge Engineering*, pages 270–272, 2010.
- [93] Zhaozheng Yin and R Collins. Belief Propagation in a 3D Spatio-temporal MRF for Moving Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [94] D Yow, B L Yeo, M Yeung, and B Liu. Analysis and Presentation of Soccer Highlights from Digital Video. *Asian Conference on Computer Vision*, 1995.
- [95] Qian Yu and G Medioni. A GPU-based implementation of motion detection from a moving platform. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008.
- [96] Y Yun and Y Liu. An improved background and foreground modeling using kernel density estimation in moving object detection. In *International Conference on Computer Science and Network Technology*, 2011.
- [97] Jiangen Zhang, Yongtian Wang, Jing Chen, and Kang Xue. A framework of surveillance system using a PTZ camera. In *IEEE International Conference on Computer Science and Information Technology*, pages 658–662, 2010.