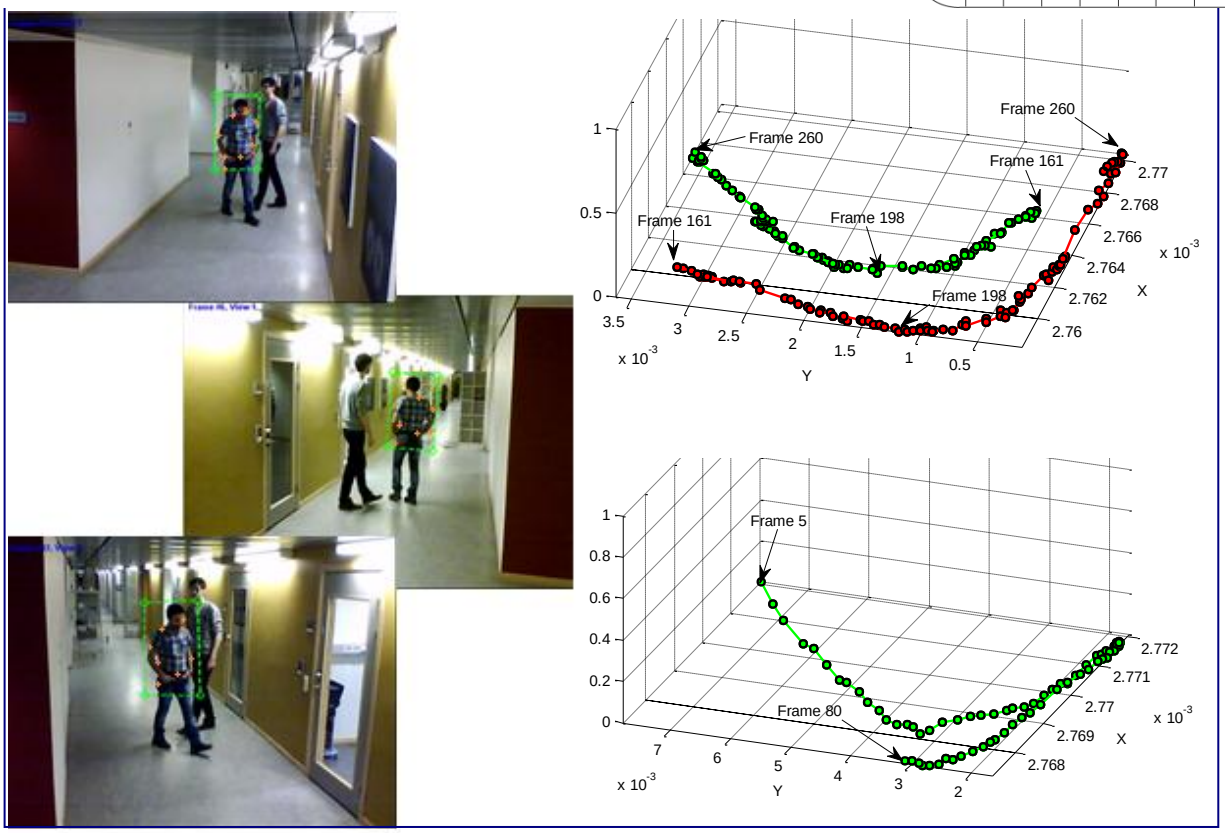
CHALMERS



Estimation of Visual Object Trajectory by using Video from Multiple Cameras

Master's Thesis

SEYED MASIH EMAMI

MORO DAVIDE

Department of Signals and Systems
Division of Signal Processing
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2012
Master's Thesis EX016/2012

MASTER'S THESIS EX016/2012

Estimation of Visual Object Trajectory by using Video from Multiple Cameras

Seyed Masih Emami

Moro Davide

Supervisor/Examiner: Irene Yu-Hua Gu

Department of Signals and Systems
Division of Signal Processing

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2012

Estimation of Visual Object Trajectory by using Video from Multiple Cameras
Master's Thesis
SEYED MASIH EMAMI
MORO DAVIDE

Master's Thesis EX016/2012
Department of Signals and Systems
Division of Signal Processing
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Department of Signals and Systems
Göteborg, Sweden 2012

Estimation of Visual Object Trajectory by using Video from Multiple Cameras
Master's Thesis
Seyed Masih Emami
Moro Davide
Department of Signals and Systems
Division of Signal Processing

Chalmers University of Technology

ABSTRACT

Many effective techniques have been proposed on intelligent vision systems for tracking video objects using multiple cameras during the last decades. Since video from multiple cameras provides rich information from different view angles and locations, object tracking using video captured from multiple cameras is usually more robust for tracking objects with full/partial occlusions and intersections. One of the previously proposed methods uses homography for estimating 3D object trajectories on the ground plane, where images from multiple cameras and a synthetic top-view plane of the scene are used. The method is rather promising.

This thesis focuses on estimating 3D object trajectories from videos captured by multiple cameras, where top-view images are not available, and cameras are not calibrated. Conventionally, estimating 3D object trajectories requires that either all cameras be calibrated, or Epipolar Geometry is used. One attractive way is to build 2D-3D relations without camera calibrations. In our study the trajectory of 3D object movement on the ground plane is derived by multiple uncalibrated cameras with the help of Epipolar Geometry without synthetic top-view scenes. This would allow cameras changing their positions or orientations during measurements. In our studies, correspondences among different camera view images from a same scene are established by using object point features from SIFT and RANSAC. This enables to establish relations between 2D-2D and 2D-3D images/object, hence, objects in 2D image planes and in the 3D world coordinate system. Within the framework of Epipolar Geometry, object positions on the ground plane are estimated up to a projective ambiguity in 3D-space (i.e. equality up to an arbitrary projective transformation in 3D-space) by minimizing re-projection error.

The methods have been tested on videos captured from 3 IP cameras (recorded by ourselves). Two types of scenarios are tested to verify our algorithm for estimating time trajectories of 3D object in the ground plane. Visual inspections of the resulted trajectories are shown to be consistent to the real movement of object.

Key Words: Epipolar Geometry, Multiple-View Geometry, Computer Vision, Visual Object Tracking, Trajectory, Point Features, SIFT, SURF, RANSAC, Geometric Transformations, Homography, Fundamental Matrix, Epipolar Constraint, Camera Matrix, Projective Geometry.

Acknowledgements

First, I would like to express my highest gratitude towards my supervisor, Professor Irene Gu for her great and kind support, patience and attention through this project. Second, I kindly and honestly appreciate Peter Strandmark for sharing his knowledge and work on the field with me. I also have to thank my parents for their support during this period. Without their unconditional love, this progress was impossible. Last but not least, I sincerely thank all my friends and the research staff in the department of Signals and Systems for providing me with such a nice and friendly environment to work.

Göteborg March 2012
Seyed Masih Emami

First of all I would like to thank Prof. Irene Gu for accepting me in the research group and giving me this great opportunity to work with her team. Great thanks go to Prof. Nicola Conci for advising me in steps of my experience, and the LLP-Erasmus program of the University of Trento, Italy for the scholarship. With a lot of pleasure I want to mention my thesis partner Masih and thank him for all the time passed with me working. Besides, my sincere thanks go to all the staff, professors and PhD students in the department of Signals and Systems at Chalmers University of Technology in Göteborg. Such persons are always a pleasure to meet in our life. Finally I want to say thanks to my family for always supporting me in many different ways and sincere thanks goes to all my friends, especially the ones in Frölunda Torg.

Göteborg March 2012
Moro Davide

Contents

Notations

Roman bold lower case letters

$\mathbf{x}^{(i)}$ Variable notation for a 2D point in the image plane of the i^{th} camera (*3x1* column vector in homogenous coordinate system)

$\mathbf{l}^{(j)}$ Variable notation for the epipolar line in the j^{th} camera image plane corresponding to $\mathrm{x}^{(i)}$ (*3x1* column vector)

$\mathbf{e}^{(i)}$ Variable notation for the epipole in i^{th} camera image plane (*3x1* column vector)

a Variable notation for an scalar value.

$\mathbf{a}$ Variable notation for a column vector.

$\mathbf{a}^{\mathrm{T}}$ Variable notation for a row vector corresponding to a.

Roman bold upper case letters

$\mathbf{X}$ Variable notation for a point in the 3D world coordinate system corresponding to its n projections in the image plane of n cameras $\{\mathrm{x}^{(i)} | i = 1, \dots, n\}$ (*4x1* column vector in homogenous coordinate system)

$\mathbf{F}_{\mathrm{ij}}$ Variable notation for the *3x3* fundamental matrix mapping 2D image points in the i^{th} camera image plane to the epipolar lines $\mathrm{l}^{(j)}$ in the j^{th} camera image plane, i.e. satisfying the epipolar constraint corresponding to the view pairs $\{i, j\}$: $(\mathrm{x}^{(j)})^T \mathbf{F}_{\mathrm{ij}} \mathrm{x}^{(i)} = 0$.

$\mathbf{P}^{(i)}$ Variable notation for the *3x4* camera matrix of the i^{th} camera.

$\mathbf{T}$ Variable notation for 2D-2D transformations (*3x3* matrix), 3D-3D transformations (*4x4* matrix), 2D-3D transformations (*3x4* matrix), and 3D-2D transformations (*4x3* matrix).

$\mathbf{H}_{\mathrm{ij}}$ Variable notation for 2D-2D planar homography transformation (*3x3* matrix) mapping 2D image points from i^{th} camera image plane to j^{th} camera image plane, i.e. $\mathrm{x}^{(i)} \overset{\mathbf{H}_{\mathrm{ij}}}{\Rightarrow} \mathrm{x}^{(j)}, \mathrm{x}^{(j)} = \mathbf{H}_{\mathrm{ij}} \mathrm{x}^{(i)}$

$\mathbf{A}_{\mathrm{n \times m}}$ Variable notation for an *nxm* matrix.

1 Introduction

*"**V**ision is the process of discovering from images what is present in the world and where it is"*
(David Marr)

Think of a day when a computerized system is able to take part in our daily vision tasks. If such dream is fully realized, a computerized system is authorized by humans to sense the environment just as a human does, so that it can sense properties of physical world like shape, color and motion. The ultimate goal of the members for the community of computer vision all over the world is to realize at least part of such dream.

In the recent years, there has been a vast amount of research by the computer vision community around the world on different topics like object detection, recognition, classification and tracking. The majority of the visual trackers in the literature are divided into 3 tracking categories: discrete feature trackers, contour tracker and region-based trackers. This categorization is based on the features used and the algorithms employed by the various visual trackers [41].

A significant consideration in the field of visual tracking is that of starting, or initializing, a tracker. Visual tracker that require less user input from a human are more practical and more likely to be deployed in real application scenarios [41].

Target initialization can be performed manually or automatically. Many significant tracking papers simply assume that the targets can be initialized (identified/detected) by some other mechanism. This assumption is typically made so that the focus of the work can be placed solely on the task of tracking the marked target through subsequent frames. When manual initialization is used, it is often assumed that all targets are present and fully visible in the first frame of the video sequence [50].

Object tracking has received considerable attention in the past few years mainly due to the wide range of its potential applications. One important application domain is advanced human–machine interfaces, where tracking, along with human motion analysis and behaviour understanding, plays a role complementary to speech recognition and natural language understanding [50]. In this context, gesture recognition, body and face pose estimation; facial expression analysis and recognition are employed in an effort to enable machines to interact more cleverly with their users and also with their environment. In all these tasks, tracking constitutes an essential part of the overall process.

Some of the other typical applications for such intelligent vision systems are in safety/security for private/public places or vehicles, visual serving by robots in industry, army, and home/business areas.

Nowadays, one open question in the state-of-the-art for the community of computer vision is how to get use of the information from multiple cameras in order to make the existing visual trackers more robust towards partial or full occlusion of the object. Recently, there has been much attention towards this issue but none of them [36, 37, 38, 39, and 40].

1.1 Problems addressed

There are a number of problems to be solved concerning a vision system. Those in which we are interested in this thesis are:

1. Discrete feature tracking of the object.
2. Recovering the true position of the object under partial/full occlusion.
3. Deriving the trajectory of the moving object in the 3D world.

In addition, one should be aware of those important challenges that a visual tracker typically faces, among those are noise, clutter (high correlation between background and some objects), illumination change, change in object scale and orientation, occlusion (partial visibility of some objects). More specifically, occlusion is one of the most important challenges that make people thinking about using more than a single camera for visual tracking. Last not least, long time occlusion would be a big challenge for a visual tracker which is still remained as an open question in the field.

1.2 Aim of this thesis work

For solving the three problems addressed in section 1.1, we choose to have 3 uncalibrated cameras with disjoint views to track a human in the scene and meanwhile deriving the trajectory of the object movement on the ground plane.

For such purpose, the object is initialized at some frames, and in the subsequent frames it has been tracked in each single view separately by using discrete point-based features. Besides, by computing the Epipolar Geometry relating the cameras to the scene and the object, the object position in 3 cameras are used to estimate its position in some arbitrarily chosen 3D world coordinate reference frame up to a reconstruction ambiguity in 3-space. By having the trajectory of object movement on the ground

plane, one might be able to understand the behaviour of the object with respect to the other objects in the scene. In the conclusion part, we mention some of the typical applications for the trajectory of the object movement in the scene.

2 Background

In this chapter, the basic methods and tools needed for the proposed methodology is presented in brief way.

The first part intends to provide a short description about interest points, their local feature descriptor, and the criterion for matching two local feature descriptors. This is the base of our work, as our tracking algorithm tracks these local point features during the video. In simple scenarios, where the object is textured, the point correspondences are suitable for tracking purpose. However, when the object is smooth or occluded, other methods may be adopted for the tracking part. Later, some other types of feature extractors are presented and for the main ones a brief comparison is done regarding their performance.

In the second part, our discussion is extended for multiple view camera analysis. All the discussions in the first part are still valid in this part, but at some places, we have done some small revisions for adopting the same concepts to the multiple-view case. Finally the geometry between the cameras is described with the main focus on epipolar geometry, epipolar constraints, camera and fundamental matrix.

2.1 Visual Object Tracking Using Single Camera

Tracking in video sequence involves matching objects in consecutive frame using some kind of information. Essentially, the method has to identify coherent relations of image information parameters between adjacent frames. The most general tracking method is based on the prediction of the object position, using the previous frames on the video sequence. Actually there are several methods, with base or not on this prediction analysis that can be classified in different categories such as global and local features. In this work, the focus would be mainly on a particular kind of local feature sets known as Speeded Up Robust Features (SURF) which is an extension to the old and well-established Scale Invariant Feature Transform (SIFT). Both SIFT and SURF generate point-based features. As its name implies, point-based features correspond to discrete image points. Besides, there would be a small part for another kind of features known as region descriptors. In contrast to point features, these features correspond to an image region. Before going through SIFT and SURF, a general description of interest point and some local features assigned to them is given.

2.1.1 Interest Point Detection

Here, a set of feature is defined as a local, meaningful representation of a target or part of it and may be not confused with a target template. A template describes the appearance of a target, regardless of the features that are being used. A good set of features may impose distinctive local characteristics on a point, such as brightness, contrast, texture. Such points can be corners, edge and point of strong change in gradient. Each of these point types will be defined later in this section.

Thus, for successful tracking, the feature set must be both descriptive and distinctive; however, it must also be flexible. A target may deform, rotate, change in illumination and change in scale. Visual trackers are desired that can continue following a target through such changes.

In order to reach a robust tracking method for objects in a single camera, there must be some a set of features invariant to different types of changes like scale, orientation and illumination changes besides noise. In short, some interest points (to be defined later on) are first detected in a frame, described by a feature vector and finally matched to another set of interest points in some other frames by some matching criterion.

To make the concept of interest points more clear, we start with corner points which are quite comprehensible. For a point with a well-defined position, a corner can be described in two ways:

1. Intersection of two edges;
2. A point for which there are two dominant and different edge directions in a local neighborhood of the point.

Most of the so called corner detection algorithms detect interest points which are more general [2].

Quality of a corner detection method is determined by its ability to detect the same corners in multiple similar images under different lightning condition, translation, scale change and rotation.

A simple approach to corner detection is to use correlation of largely overlapping patches around each pixel of the image. This approach is very computationally expensive and suboptimal. An alternative is the method proposed by Harris and Stephans that is an improved version of a method by Moravec [49].

Moravec defined the similarity measure between two image patches as SSD (Sum of Squared Difference) of the image patches. Clearly, a lower SSD shows higher similarity between two patches. The corner strength is defined as the smallest SSD between the patch and its neighbours (horizontal, vertical, two diagonals) and the corner is defined as a point in which SSD is locally maximized.

A drawback for this method is that it is not isotropic. If an edge is present which is not in the direction of the neighbours, then it will not be detected as an interest point.

As a solution to this problem, Harris and Stephans used the local auto-correlation function of a signal; where the local auto-correlation function measures the local changes of the signal with patches shifted by a small amount in different directions [11].

Given a shift $(\Delta x, \Delta y)$ and a point (x, y), the auto-correlation function is defined as,

$$c(x,y) = \sum_{\mathbf{W}} [\mathbf{I}(x_i, y_i) - \mathbf{I}(x_i + \Delta x, y_i + \Delta y)]^2 \tag{2.1}$$

Where $\mathbf{I}(.,.)$ denotes the image function and (x_i, y_i) are the points in the window **W** (Gaussian[1]) centred on (x, y).

The shifted image is approximated by a Taylor expansion truncated to the first order terms,

$$\mathbf{I}(x_i + \Delta x, y_i + \Delta y) \approx \mathbf{I}(x_i, y_i) + [I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta \mathrm{y} \end{bmatrix} \tag{2.2}$$

where $I_x(.,.)$, $I_y(.,.)$ denote the partial derivative in x, y directions respectively.

Substituting approximation (2.2) in (2.1) yields,

$$\begin{aligned} c(x,y) &= \sum_{\mathbf{W}} [\mathbf{I}(x_i, y_i) - \mathbf{I}(x_i + \Delta x, y_i + \Delta y)]^2 \\ &= \sum_{\mathbf{W}} \left(\mathbf{I}(x_i, y_i) - \mathbf{I}(x_i, y_i) - [I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta \mathrm{y} \end{bmatrix} \right)^2 \\ &= \sum_{\mathbf{W}} \left([I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta \mathrm{y} \end{bmatrix} \right)^2 \\ &= [\Delta x \quad \Delta \mathrm{y}] \begin{bmatrix} \sum_{\mathbf{W}} \left(I_x(x_i, y_i)\right)^2 & \sum_{\mathbf{W}} I_x(x_i, y_i) I_y(x_i, y_i) \\ \sum_{\mathbf{W}} I_x(x_i, y_i) I_y(x_i, y_i) & \sum_{\mathbf{W}} \left(I_y(x_i, y_i)\right)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \mathrm{y} \end{bmatrix} \\ &= [\Delta x \quad \Delta \mathrm{y}] \mathbf{C}(x,y) \begin{bmatrix} \Delta x \\ \Delta \mathrm{y} \end{bmatrix} \end{aligned} \tag{2.3}$$

[1] For clarity in exposition, the Gaussian weighting factor $e^{-(x^2+y^2)/(2\sigma^2)}$ has been omitted from the derivation.

Where matrix $\mathbf{C}(x, y)$ captures the intensity structure of the local neighbourhood.
Let λ_1, λ_2 be the eigenvalues of matrix $\mathbf{C}(x, y)$. The eigenvalues form a rotationally invariant description. There are three cases to be considered:

1. If both λ_1, λ_2 are small, so that the local auto-correlation function is flat (i.e., little change in $c(x, y)$ in any direction), the windowed image region is of approximately constant intensity.

2. If one eigenvalue is high and the other low, so the local auto-correlation function is ridge shaped, then only local shifts in one direction (along the ridge) cause little change in $c(x, y)$ and significant change in the orthogonal direction; this indicates an edge.

3. If both eigenvalues are high, so the local auto-correlation function is sharply peaked, then shifts in any direction will result in a significant increase; this indicates a corner.

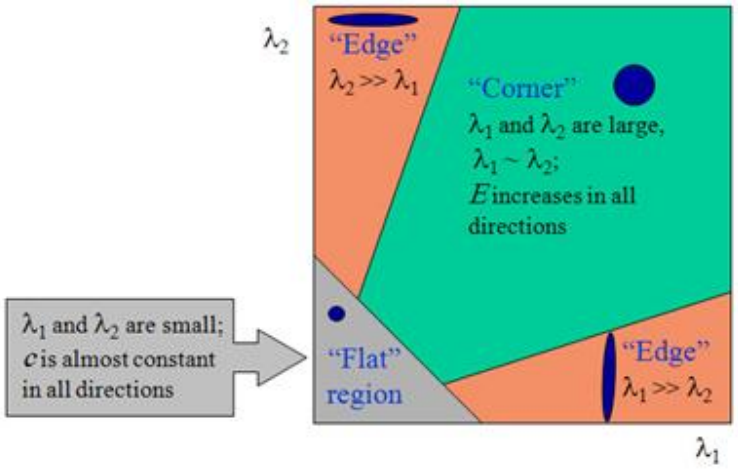


Figure 2.1 *Classification of image points using eigenvalues of* **C**.

Some proposed measures of cornerness for a point (x, y) can be:

$$Det(\mathbf{C}) = \lambda_1\lambda_2$$

$$Trace(\mathbf{C}) = \lambda_1 + \lambda_2$$

$$Cornerness1 = Det(\mathbf{C})/Trace(\mathbf{C})$$

$$Cornerness2 = Det(\mathbf{C}) - k(Trace(\mathbf{C}))^2, k = 0.04 - 0.06 \quad (2.4)$$

The main drawback for this method is that it is not invariant towards scale change. As a solution, use of derivative of Gaussian is proposed. Harris-Laplacian which gets use

of this solution is the most robust corner detector towards affine changes (i.e. changes by scale, translation, or rotation).

2.1.2 Interest Point Description by SIFT

The Scale Invariant Feature Transformation or SIFT is a well-established algorithm for extracting a set of features from an image point. This kind of approach, presented for the first time in [10], is widely used in various kinds of applications range from robotics to image retrieval and object recognition, visual mapping and analysis, image stitching, motion tracking and 3D reconstruction. It is considered to be one of the most powerful candidates for feature detection, extraction and matching.

The principal difference between SIFT and other feature extractors is that the features are local and invariant to affine changes. At the same time, they are robust towards changes in illumination and image noise and insensitive to a wide range of image transformations and change of view in 3D space.

Here, the methodology to detect and extract some features from image points is opened up in more detail, since the main focus of this thesis is on these features.

As the first step, the image **I**(*x,y*) should be filtered by a Gaussian Filter of variance σ, which creates a "scale-space" of the image defined as a function of a position (*x,y*) and a scale *σ*:

$$\mathbf{L}(x, y, \sigma) = \mathbf{G}(x, y, \sigma) * \mathbf{I}(x, y) \tag{2.5}$$

where $*$ is the convolution operator and **G**(*x,y,σ*) is:

$$\mathbf{G}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2.6}$$

The most stable corner points are obtained by localizing the maxima and minima of a Laplacian of the Gaussian or LoG. It has been demonstrated that the Difference of Gaussian, DoG, calculated as a difference of two adjacent scale, separated by a constant *k*, is a good approximation of the LoG operator, and for this reason adopted by SIFT as follows:

$$DoG(x, y, \sigma) = (\mathbf{G}(x, y, k\sigma) - \mathbf{G}(x, y, \sigma)) * \mathbf{I}(x, y)) = \mathbf{L}(x, y, k\sigma) - \mathbf{L}(x, y, \sigma) \tag{2.7}$$

So, to find the local extremes, next step is to create two pyramids of the image, see Figure 2.2. For each octave of the scale-space, the original image is convoluted with a different set of Gaussian filters, **G***(x,y,σ)*. Then, each two adjacent images are subtracted from each other to form the **D***(x,y,σ)*. One octave of the scale space

corresponds to a set of *s* images, of the same dimension but filtered out by doubling σ each time. The optimal choice to sample the scale space function at three different levels per octave corresponds to imposing $k \pm 1.26$.

Local extremes have been searched for each level of the pyramid of *DoG*. Each point is compared by the *(n x n)* adjacent pixels on the same scale and by the *(n x n)* from the other two adjacent scales, top and lowers (see Figure 2.3 in which *n* is equal to 3).

Successively the research is focused on finding only the best key points, therefore the objective is to discard those points with low contrast and so more sensible to the image noise, and those placed on edges. So, unsuitable points are removed, because their positions are ill-defined.

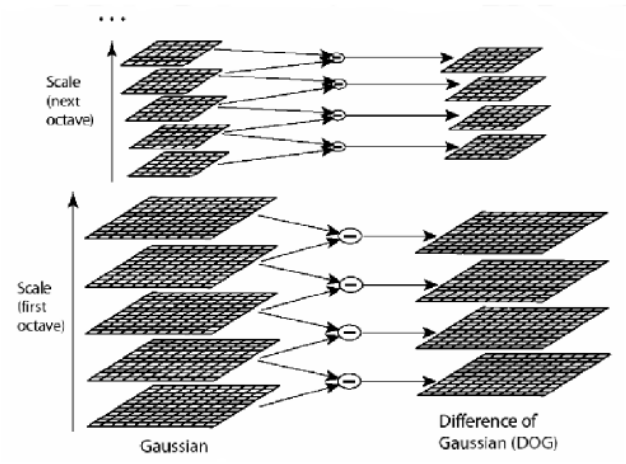


Figure 2.2: *Difference of Gaussian in pyramid of images. The figure is taken from [10].*

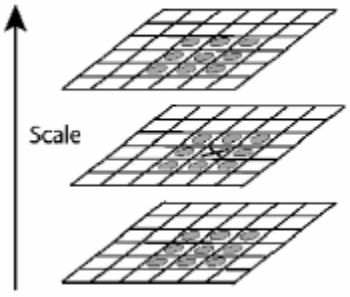


Figure 2.3: *The central point (**x**) is compared with all the adjacent pixels in the top, same and lower scale. The figure is taken from [10].*

Once the key points are defined with their coordinates and related scale, one dominant orientation needs to be assigned to each feature point, to achieve the

rotational invariance. This is done by sampling the magnitude *m(x,y)* and the orientation *υ(x,y)* of the gradient at different sample points in a neighbourhood around the feature point.

$$m(x,y) = \sqrt{((\mathbf{L}(x+1,y) - \mathbf{L}(x-1,y))^2 + ((\mathbf{L}(x,y+1) - \mathbf{L}(x,y-1))^2}$$
$$\vartheta(x,y) = atan2(\mathbf{L}(x,y+1) - \mathbf{L}(x,y-1) \,/\, \mathbf{L}(x+1,y) - \mathbf{L}(x+1,y)) \quad (2.8)$$

These samples are then sorted out into a histogram with 36 bins, in a way to cover the full 360 degrees around the point. The highest bin is then chosen as the principal direction. Additional peak, with 80% of the maximum value becomes as a principal direction for additional feature points at the same location. So in some cases there will be multiple features associated to each image location with the same position and scale, but different orientation.

At the end, some key points are extracted with specific coordinates, scale and canonical orientation. The last step is to generate a local descriptor for the features in a neighbourhood around each key point. The creation of a descriptor, illustrated in Figure 2.4, is done using the pre-calculated data during the orientation assignment. At the beginning, a fixed area (generally *16x16* samples) is considered around the key point pre selection in the pyramid of Gaussian images **L**(*x,y,σ*).

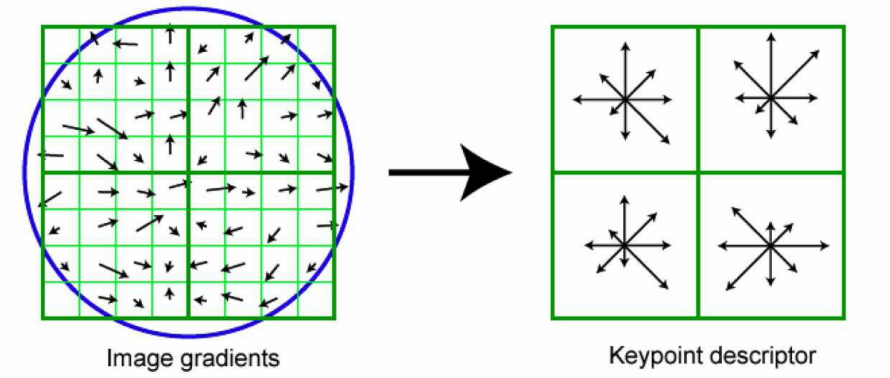


Figure 2.4: *Local Descriptors. A Descriptor has 2x2 dimensions, constituted from a set of 8x8 of the image* **L**. *The figure is taken from [10].*

To ensure the rotation invariance, the coordinates and the direction of the local gradients are rotated with respect to the key point orientation. For all surrounding points, the magnitude and orientation values are calculated and weighted by a Gaussian window. More emphasis is on the samples near the centre of the descriptor and this area is divided into *4x4* sub-regions. Afterwards, the orientation of the singular samples around the key point, are gathered into a histogram with 8 bins

corresponding to 8 directions. The bins of all *4x4=16* histograms form the 128 element feature descriptor vector. Each element of this vector corresponds to the value of a bin in the orientation histogram of the relative *4x4* sub-region around the key point. So at the end we have a single location *(x,y)* in the source image, with an associated orientation and scale, that is visually distinct from its surroundings.

Input: Image **I**.

Output: Interest points positions and their 128-elements descriptor vector.

1. Detect/Identify key points (Visually distinct from its surroundings)

a. Compute a pyramid of Gauss-filtered images, organized into octaves of (*S+1*) images. (In practice, *S* could be 3) Each image is smoothed by a factor of *k* more than the image below. So an octave is a set of Gaussian-convolved image, $\mathbf{I}_1 ... \mathbf{I}_s$ representing a doubling of the scale parameter for **G** between $\mathbf{I}_1$ and $\mathbf{I}_s$; Images in the next octave are sub-sampled and stored at ½ resolution of the previous octave

b. Compute a pyramid of DoG-filtered images by getting the difference between two adjacent Gaussian images.

c. Locate extrema of DoG pyramid (x_i, y_i, ρ_i) in which ρ_i is scale. Find all pixels that correspond to extrema of DoG, i.e. apply a filter on each difference image.

d. Refine location of DoG extrema $(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$ through a quadratic least square fit and Euclidean distance.

e. Prune all extrema that are weak or that correspond to edges.

f. Assign orientation to extrema $\mathbf{p}_i = (x_i', y_i', \rho_i', \theta_i)$ i.e. compute the histogram of orientations.

2. Build feature vectors for the identified key points from step 1.

g. Compute gradients in 16×16 pixel patch of image $\mathbf{I} * \mathbf{G}_{\sigma_i}$ (Gaussian-smoothed image at the scale of key point) centered at (x_i, y_i).

h. Compute gradient orientation relative to key point orientation.

$$\theta(x, y) = \tan^{-1}\left[\frac{\partial(\mathbf{I} * \mathbf{G}_{\sigma_i})}{\partial \mathrm{y}} \Big/ \frac{\partial(\mathbf{I} * \mathbf{G}_{\sigma_i})}{\partial \mathrm{x}}\right] - \theta_i$$

i. Compute orientation histogram of each 4×4 pixel block (containing 8 bins, each covering 45˚). In other words, define an "accumulator" variable for each of the orientations in each of the 16 histograms (8 bins * 16 histograms = 128 accumulators totally)

j. For each pixel, calculate the pixel's contribution to each of the accumulators variable from step c and form a vector with the length of 128 (8 bin * 16 blocks = 128 accumulators). This vector represents the SIFT descriptor or feature vector.

k. Normalize the feature vector.

l. Clamp all the elements of the feature vector at 0.2.

m. Re-normalize the feature vector.

Algorithm 2.1. *SIFT descriptors*

2.1.3 Interest Point Description by SURF

The Speeded Up Robust Transform or SURF feature extractor is one of the most recent algorithms for point feature extraction. The same as SIFT, SURF is an affine-change invariant interest point detector and descriptor, first presented by Herbert Bay in 2006. It has slight differences with SIFT and Harris. It is more complex and partly inspired by the SIFT descriptor while performing several times faster. In other words, it can be considered as an evolution of SIFT. In short, SURF relies on integral images for image convolutions, built upon the strengths of the leading existing interest point detectors and descriptors, using a Hessian matrix-based measure for the detector and a distribution-based descriptor [15].

For identifying the key points in the image, SURF utilizes an intermediate representation of the image i.e. the integral image. In this way the performance is highly increased and the features are extracted much faster.

The basic difference of SURF and SIFT is the way to calculate the convolution of the image. There is one special situation where the computation does not depend on the filter size; namely, where the filter is rectangular and flat (i.e., with constant coefficients). One method in fact convolves an image with such rectangular flat filters of arbitrary size, and that performs the convolution in a time that is independent of the size of the filter.

Give an image **I** and one image point *(x,y)*, the integral image $\prod(x, y)$ is the sum of the pixel intensity values inside the rectangle between *(x,y)* and the origin of the image on the top left corner as stated in equation 2.9.

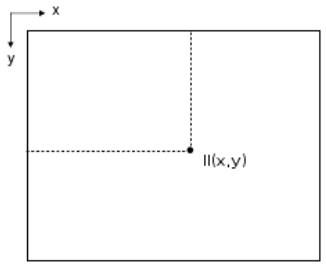


Figure 2.5: *The integral image*

$$\prod(x, y) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} \mathbf{I}(x, y) \tag{2.9}$$

This is the definition of the accumulator sum array, where the value of each pixel in the accumulator is the sum of all pixels values in the source that are in the rectangle that is above and to the left of the pixel.

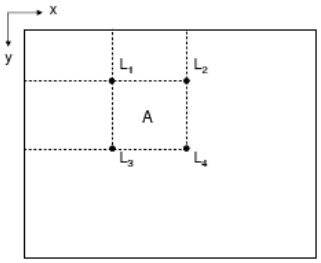


Figure 2.6: *Calculation of a window area A, in the integral image*

With this new integral image, it is very fast to calculate the sum of the intensity values in any window inside an image. For example, for the rectangle *A* in Figure 2.6 the sum of the intensity of its pixels is calculated by:

$$\prod = L_1 + L_4 - L_3 - L_2 \tag{2.10}$$

where L_i are the integral values corresponding to the coordinates of the relative point. The great advantage of this method is that the time for calculation is independent of the dimensions for the area of interest. Therefore, in this way the convolution of filters with different sizes is done much faster compared to SIFT and Harris.

Another important aspect for SURF is the hessian matrix which is much more accurate than other kind. More precisely, this technique defines blob-like structures where the determinant of the matrix is maximized.

In the general case of a continuous function of two variables *f(x,y)*, the Hessian matrix is the partial derivate of the function itself:

$$H\big(f(x,y)\big) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial^2 x} & \frac{\partial f(x,y)}{\partial x \partial y} \\ \frac{\partial f(x,y)}{\partial x \partial y} & \frac{\partial f(x,y)}{\partial^2 y} \end{bmatrix} \tag{2.11}$$

and its determinant is:

$$\left|H\big(f(x,y)\big)\right| = \frac{\partial f(x,y)}{\partial^2 x}\frac{\partial f(x,y)}{\partial^2 y} - \left(\frac{\partial f(x,y)}{\partial x \partial y}\right)^2 \tag{2.12}$$

Since the determinant coincide with the product of auto eigenvalues of the matrix, it is possible to understand if some points are an extrema of the function by simply

looking at the sign of the determinant; if is negative, the eigenvalues have different sign so the point is not a local extrema, and if positive, the point is a local minimum or maximum.

By substituting the function *f(x,y)* with the integral image $\prod(x, y)$, the partial derivatives are obtained from the convolution on filters.

The second order normalized Gaussian filter allows us to analyze the image space in different scale factors. In fact, given a point *(x,y)* of an image **I**, the hessian matrix would be:

$$\mathcal{H}(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{2.13}$$

where $L_{xx}(x, \sigma)$ is the convolution of the second order derivative of the Gaussian $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image on the point x, and similarly for $L_{xy}(x, \sigma)$ and $L_{yy}(x, \sigma)$. As in SIFT, we may evaluate the determinant from different scale and covariance Gaussian convolution.

Another benefit in fact is to apply the box filter in increasing dimensions, instead of decreasing the reference image resolution, like in Figure 2.7.

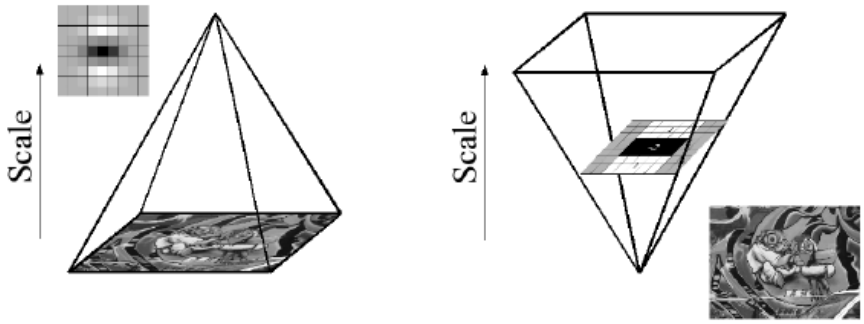


Figure 2.7: *Calculation of a window area A, in the integral image.*

The advantages of this method are evident. In SURF, the scale of the filter is changed and not the image resolution, while the computational cost remains constant. Also, there is a possibility to compute different levels of the pyramid simultaneously. As in SIFT, the scale-space is divided into a number of octaves, and each octave represents a filtered version of the image. The filter dimensions grow larger by increasing the number of octave.

In total, one octave has a factor 2 of scales and each divided into a constant number of levels. The construction of the scale-space starts by applying a *9×9* filter, and continues by *15×15*, *21×21*, *27×27*, etc. At the same time, the covariance matrix starts from 1.6 and continues to 3.2.

There are basically three steps to localize a key point in SURF:

1) Filter with a threshold value ;
2) Each key point is compared with the 8 neighbouring pixels of the original image and with the 9 neighbouring pixels in the upper and lower scale: if the intensity is the maximum or minimum, point is maintained;
3) Interpolate the points near to each other to improve the accuracy.

The process of extracting a descriptor for key points in SURF is similar to SIFT. In SURF, descriptor vector is identified by the responses from Haar filters and not from the gradient. For each interest point, a dominant direction is extracted, and inside a window centred on the same point, a 64-element descriptor vector is extracted. This vector represents a point in the feature space, in which the dominant orientation is suitably assigned. When the dominant direction is extracted, the next process consists of constructing a square region centred on the interest point and oriented along the specified dominant direction.

The first step for extracting the descriptor is to find the direction of key points and form a circular region around that point. The response of the wavelet is calculated and then weighted with a low-pass Gaussian filter to give more importance to the response close to the point of interest. The total orientation is obtained by summing over all the responses obtained in different directions. Afterwards, a window is created on the key point and divided into *4×4* sub-regions which in turn are divided by a regular *5×5* grid.

At each grid point, filters are applied along the directions x and y and for each block, a vector is constructed as follows:

$$\mathbf{v} = \left(\sum |d_x|, \sum |d_y|, \sum |d_x|, \sum |d_y|\right) \tag{2.14}$$

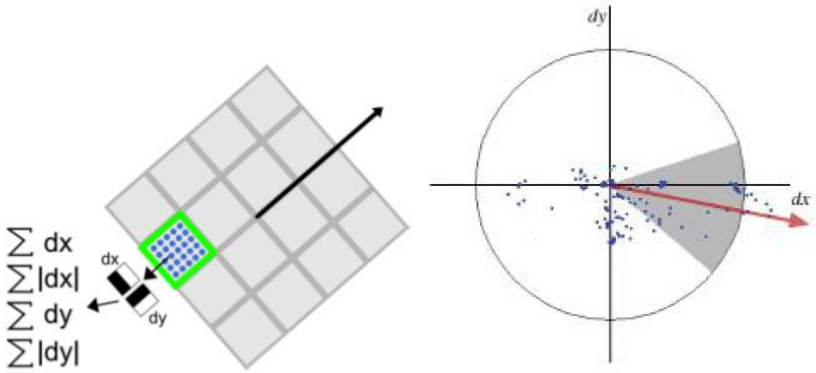


Figure 2.8: *Dominant orientation selection based on the response from Haar filters.*

This results in a vector of 64-element descriptor vector given by 4 dimensions for each *4×4* sub-regions. In addition to the default descriptor size for SURF which is 64, there is also the possibility to extend to 128-element descriptor vector, like in SIFT.

2.1.4 Interest Point Matching

By SIFT/SURF, some key/interest points along their features are detected and extracted from two images of a scene **I**, **I′**. Afterwards, by some similarity measure, these key points are matched between the two images. Often, many of these correspondences are incorrect, mainly due to 2 types of error:

- False positive matches: matching returns a false correspondence.
- False negative matches: matching fails to detect a true correspondence.

The goal is to minimize the number of false positive and negatives across wide range of imaging conditions. As mentioned earlier, SIFT features are invariant to a certain types of image transformations (i.e. affine transformations).

When all the key points and their descriptors are computed from both images, the nearest neighbour is considered as the best match. The utilized distance is the Euclidean distance. To reduce the number of false positives, matching with a distance smaller than a threshold is considered; otherwise the match is considered unreliable and is rejected. The recommended matching criterion is summarized in algorithm 2.2 and also a simple case is shown in figure 2.9.

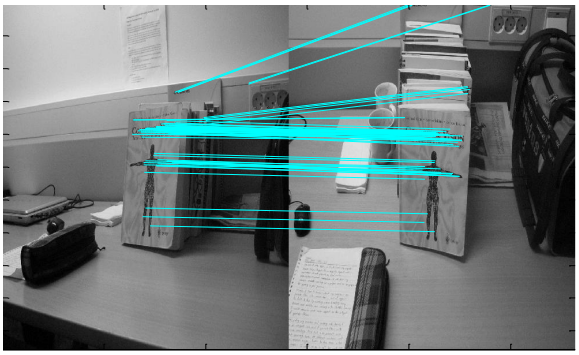


Figure 2.9: *Matched key points with SIFT/SURF descriptors.*

Input:

- Two set of interest points and their 128-element descriptor vectors.
- Matching Threshold.

Output: Value of matching for each Point Feature Descriptor for each image.

1. Find the key points of each given image and calculate SIFT descriptor vectors corresponding to each key point.
2. Match these key points with the following criteria:

$$\Phi = \frac{\cos^{-1}(\mathbf{f}_{\mathrm{i}}.(\mathbf{f}'_{\mathrm{j}*})^{\mathrm{T}})}{\cos^{-1}(\mathbf{f}_{\mathrm{i}}.(\mathbf{f}'_{\mathrm{j}**})^{\mathrm{T}})} < 0.6$$

i, j refer to the two images which are being matched.
* means the 1st closest descriptor in the sorted array of vector dot products.
** means the 2nd closest descriptor in the sorted array of vector dot products.

Algorithm 2.2. *SIFT/SURF Matching Criterion*

2.1.5 2D Transformation Models

Given a set of point correspondences $\left\{\mathbf{x}_i^{(j)} \leftrightarrow \mathbf{x}_i^{(k)}, i = 1, \ldots, n\right\}$, the goal is to define a mapping function $\mathbf{T}$ which maps all $\mathbf{x}_i^{(j)}$ to $\mathbf{x}_i^{(k)}$. There are a number of candidates for such mapping function. Before introducing these candidates or so called transformation models, one must be aware of homogenous and inhomogeneous coordinate systems. Refer to appendix A for more detailed explanation.

Such transformation models simply change the coordinates of some 2D image points. Each of these image points are expressed as a 2-element vector in Euclidean space or 3-element vector in projective space. A transformed set of points will result in a different shape, considering those points as the vertices of the object boundary.

The theory of linear algebra demonstrates that 2D linear transformations can be represented by *2×2* matrices in Euclidean space and 3×3 matrices in projective space. In this thesis, the main focus will be on the projective space, hence 2D linear transformations are expressed as *3×3* matrices and the points are expressed in homogeneous coordinates.

The general form of a 2D linear transformation is:

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ \alpha_7 & \alpha_8 & \alpha_9 \end{bmatrix} \tag{2.15}$$

Where all α_i are scalar values.

In homogeneous coordinates, a set of points are expressed in matrix format as below:

$$\mathbf{S}_{3\times n}^{(i)} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ y_1 & y_2 & y_3 & \dots & y_n \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \tag{2.16}$$

And T should be estimated, satisfying the following equation:

$$\mathbf{S}^{(j)} = \mathbf{T}\mathbf{S}^{(i)} \tag{2.17}$$

Later, in section 2.3.1.1 we explain how to estimate T given $\mathbf{S}^{(i)}$ and $\mathbf{S}^{(j)}$. Here, the aim is to just give an overview of different types of 2D transformation models. We start with 3 simple transformation models: translation, rotation and scaling. All of these transformations preserve parallel lines and ratio of distances in addition to angles, look at figure 2.10.

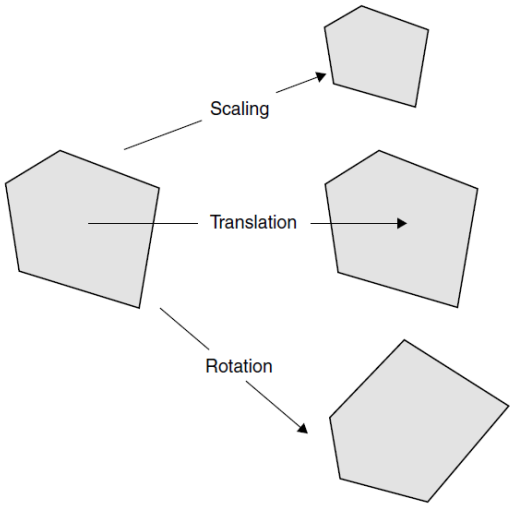


Figure 2.10: *Example of 3 simple transformations which preserve the shape of the object.*

An affine transformation is a transformation which preserves parallel lines and ratios of distances, but not angles and the distances. The 2D affine transformation in homogeneous coordinates takes the general form:

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

By such form, an affine transformation could transform a set of points in many different ways, as summarized in table 2.1. Some typical results are shown in figure 2.11 for more clarity.

Table 2.1 *Possible subsets of a 2D affine transformation in homogeneous coordinates*

Transformation	α_1	α_2	α_3	α_4	α_5	α_6
Translation by (x,y)	**1**	**0**	x	**0**	**1**	y
Rotation by θ	$\cos\theta$	$\sin\theta$	**0**	$\sin\theta$	$\cos\theta$	**0**
Uniform scaling by s	s	**0**	**0**	**0**	s	**0**
Vertical shear by s	**1**	s	**0**	**0**	**1**	**0**
Horizontal shear by s	s	**0**	**0**	s	**1**	**0**

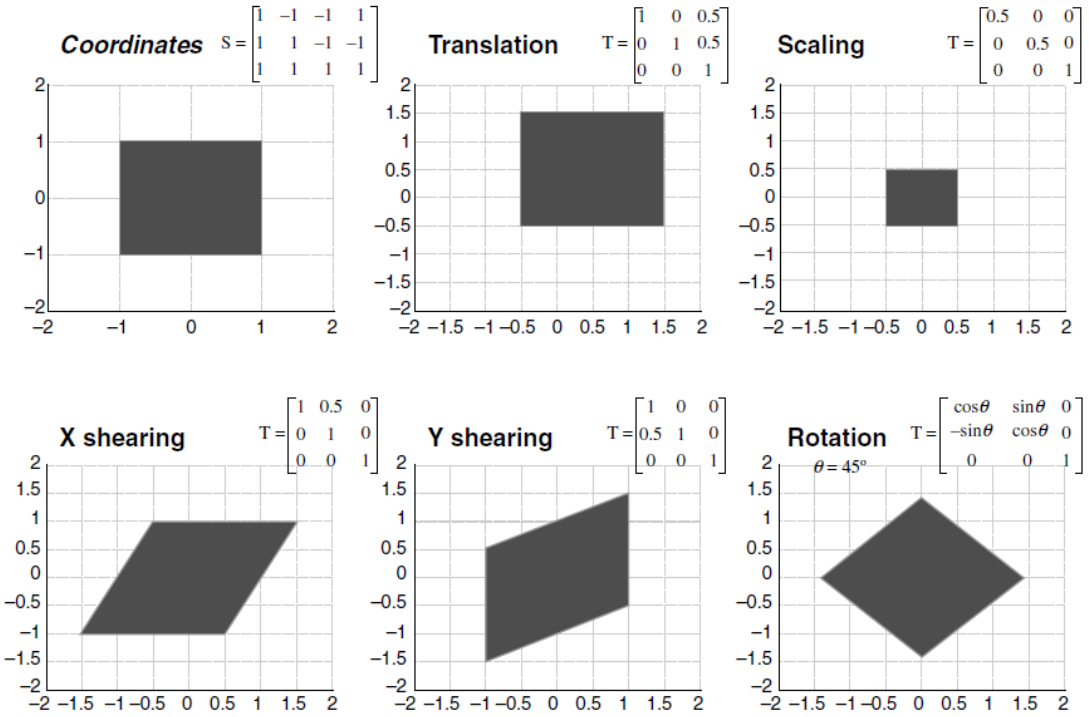


Figure 2.11: *Visualizing the result of transformations in table 2.1 on a typical set of points.*

The most general form of a 2D linear transformation is named as 2D projective transformation. It takes the general form:

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ \alpha_7 & \alpha_8 & 1 \end{bmatrix} \tag{2.19}$$

The advantage of projective transformation over affine one is the capability to model the perspectivity when an image of a 3D object is captured from a camera in with a corresponding perspective mapping. This transformation does not preserve parallel lines, angles and the ratio of distances but only straight lines. In figure 2.12, a simple example of such transformation is shown.

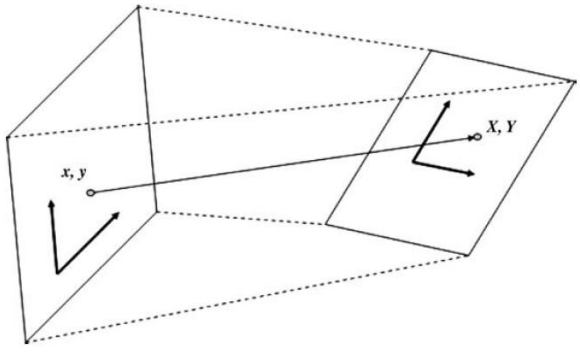


Figure 2.12: *A simple example for 2D projective transformation.*

In table 2.2, a summary of different 2D linear transformations is given with their relative degree of freedom. The higher the degree of freedom, the more complicated the model will be.

Table 2.2 *Summary of 2D Linear Transformations*

Transformation	**Permissible operations**	**Formula**	**DOF**	**Minimum Number of Points**
Translation	translation	$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	2	1
Scaling (Isotropic)	Scaling	$\mathbf{T} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$	1	1
Rotation	Rotation	$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	1	1
Euclidean	Translation, Rotation	$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	3	2
Similarity	Translation, Isotropic Scaling, Rotation	With reflection (Reflective): $\mathbf{T} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ -s\sin\theta & -s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$ Without reflection (Non-reflective): $\mathbf{T} = \begin{bmatrix} s\cos\theta & s\sin\theta & t_x \\ -s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	4	2
Affine	Translation, Non/isotropic Scaling, Rotation, Shearing	$\mathbf{T} = \begin{bmatrix} s\cos\theta & -\sin\theta & t_x \\ s\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	6	3 (non-collinear)
Projective	Translation, Non/isotropic Scaling, Rotation, Shearing, Perspective mapping	$\mathbf{T} = \begin{bmatrix} s\cos\theta & -\sin\theta & t_x \\ s\sin\theta & \cos\theta & t_y \\ h_x & h_y & 1 \end{bmatrix}$	8	4 (non-coplanar)

2.1.6 RANSAC

As SIFT/SURF gives rise to point-based features, it is much sensitive to different changes in the image data (changes in brightness, scale, orientation and image noise). As a result, some or many of the point matches by SIFT/SURF may be incorrect, so a method is needed to eliminate false positive and false negative matches.

RANdom SAmple Consensus or RANSAC tries to find the largest set of point correspondences consistent with a specific model and excluding as many incorrect point matches as possible from the set of point correspondences. Such incorrect point matches are known as outliers.

In linear algebra, the most popular method for fitting a line to several points is the least square method. However the least square has a problem: if few points in the set are completely irrelevant to the underlying model which has generated these points, the estimated line would not be the optimal fit to these set of points.

Interestingly, RANSAC can easily solve such problem. As mentioned briefly, RANSAC is an iterative method to estimate the parameters of a model complying with as many data points as possible. The output of all the iteration is correct by a predefined probability. This probability increases as more data points are found to be consisting with a model within future iterations. The algorithm was first published by Fischler and Bolles in 1981. The outliers may have roots in the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data and the underlying model.

The advantage of RANSAC is that no accumulator array is needed and so the algorithm can be more space efficient and potentially less prone to the choice of bin size. The disadvantage is that many more hypotheses may need to be generated and tested than those obtained by finding peaks with for example least square method.

RANSAC start by selecting (at random) a subset of k correspondences, which is then used to compute an initial estimate for p. p is the residuals for the full set of correspondences (i.e. the distance of each data point to the model).

Then, the number of inliers that are within a predefined distance/threshold is counted, i.e. $(residual \leq threshold)$. The random selection process is repeated S times and the point correspondence set with the largest number of inliers is chosen as a final solution. To ensure that the random sampling has a good approximation of a correct

set of inliers, a sufficient number of trials *S* must be done, the underlying model must be suitable defined and the fitness metric for measuring the distance of a data point to the model.

Let *p* be the probability that any given correspondence is valid and *P* be the total probability of success after *S* trials. The likelihood in one trial that all *k* random samples are inliers is p^k. Therefore, the likelihood that *S* such trials will all fail is:

$$1 - P = (1 - p^k)^S \tag{2.20}$$

and the required minimum number of trials is [16]:

$$S = \frac{log(1-P)}{log(1-p^k)} \tag{2.21}$$

Input: Set of *n*+*k* hypothetical matched points (where *k* are possible number of outliers), a list of possible models to select (e.g. table 2.2).

Output: Set of best *n* inliers, best inliers model and corresponding total error value.

1. Randomly select *n* values from data as the hypothetical inliers set.
2. Find the parameters of a model that fits these *n* values and name it as inliers model.
3. For all the other values not in the hypothetical inliers set, if they have an error lower than *t* when fitted to inliers model, add them to hypothetical inliers set (a typical value for *t* is *0.01*).
4. If the number of elements in the hypothetical inliers set is greater than *d*, recalculate the model parameters that fit these data (*d* is the minimum number of inliers needed for a model)
5. Measure the total error that this model introduces regarding all the data that it has. If this error is less than the best error found till now, replace the best model parameters and corresponding error value by the new values.
6. Continue doing the first 5 steps for *(k-1)* times where *k* is the number of iterations (a typical value for *k* is *1000*).
7. Present the best inliers model, inliers set and corresponding total error value.

Algorithm 2.3 *RANSAC*

2.1.7 RAMOSAC

In the state of the art, there are many different variants of RANSAC method. Each of these variants may be good for specific data sets and imposed conditions. They basically aim for increasing the performance (in terms of minimizing the total error produced by fitting a model to the data set) when the data is very sparse or contains a large amount of outliers. One of the recent variants proposed in [17] is called as Random Model and Sample Consensus (RAMOSAC). The main advantage of

RAMOSAC over RANSAC is the scoring scheme used for rejecting highly improbable models.

In summary, RAMOSAC goes through an iterative process and each time randomly selects a motion model number. At the same time, it randomly selects a set of point matches and fits a model to the data points, classify as inliers those point matches which lie within a specific threshold and add those to the set of consensus points. Finally, by using a probability scheme, a score is assigned to each model according to the formula

$$s = |C| - \frac{\lambda}{r}\sum_{i=1}^{r} \|T(v_i) - v_i\| + \varepsilon n_m$$

and the model with the highest score is chosen as the best model. The main difference between RAMOSAC and RANSAC is the last two terms in the formula above. These two terms improve the accuracy of inlier selection process.

However, in [18] it has been shown that by using both object and background point-based features, the robustness of RAMOSAC could be increased much more. In this thesis, we are only interested in using the scoring scheme of RAMOSAC and by using more advanced models derived from the epipolar geometry, we proposed another variant which is much similar to RAMOSAC and will be described in section 3.1.3.

2.1.8 HoG

In addition to point-based features as a local descriptor for a target, we here bring up another set of descriptive features as a global histogram-based descriptor for the target, specifically for characterizing human images named as Histogram of Gradients or HoG [19, 20 and 21].

There are a number of variants for these set of features such as R-HoG, C-HoG, Bar HoG, and Center Surround HoG. Each of these variants will be described in short later on in this section.

The basic idea in HoG is to count the number of occurrences for gradient orientations in different directions in the localized portions of an image. The summary of algorithm for HoG and its different variants is given in algorithm 2.4.

Common Initial steps:

1. Gamma normalize each color channel of the input image by means of square root gamma compression (photon noise on CCD detectors $\propto \text{intensity}^2$). Simply, replace each pixel value with its square root value.
2. For each color channel, compute gradients by convolving with [-1,0,1] and $[1,0,-1]^T$. Select the channel with the largest gradient magnitudes in each case and calculate total gradient magnitude matrix:

$$\mathbf{I}' = \sqrt{\mathbf{I}_x^2 + \mathbf{I}_y^2}$$

3. If using R2-HoG, smooth the image using Gaussian kernel of width ρ. Compute derivatives $I_{ij}, i, j \in \{x, y\}$ and calculate the dominant orientation by the following formula:

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2I_{xy}}{I_{yy} - I_{xx}}\right)$$

Algorithm 2.4. *HoG, Common initial steps*

Descriptor Computations: For each position of the block on the image region

1. If using R-HoG
   a. Divide the $\varsigma\eta \times \varsigma\eta$ image region centered on the point into cells
   $$\varsigma \times \varsigma = Number\ of\ cells\ in\ each\ block$$
   $$\eta \times \eta = Number\ of\ pixels\ in\ each\ cell$$
   b. Apply a Gaussian window with $\sigma = 0.5 \times \varsigma\eta$ to image gradients in block
   c. Create a $\varsigma \times \varsigma \times \beta$ spatial and orientation histogram where:
   $$\beta = Number\ of\ Orientation\ Bins$$
   d. For each pixel in the block, use trilinear interpolation to vote into the histogram using gradient magnitudes:

$$h(x_1, y_1, z_1) = h(x_1, y_1, z_1) + w\left(1 - \frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right)\left(1 - \frac{z - z_1}{b_z}\right)$$

$$h(x_1, y_1, z_2) = h(x_1, y_1, z_2) + w\left(1 - \frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right)\left(\frac{z - z_1}{b_z}\right)$$

$$h(x_1, y_2, z_1) = h(x_1, y_2, z_1) + w\left(1 - \frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right)\left(1 - \frac{z - z_1}{b_z}\right)$$

$$h(x_2, y_1, z_1) = h(x_2, y_1, z_1) + w\left(\frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right)\left(1 - \frac{z - z_1}{b_z}\right)$$

$$h(x_1, y_2, z_2) = h(x_1, y_2, z_2) + w\left(1 - \frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right)\left(\frac{z - z_1}{b_z}\right)$$

$$h(x_2, y_1, z_2) = h(x_2, y_1, z_2) + w\left(\frac{x - x_1}{b_x}\right)\left(1 - \frac{y - y_1}{b_y}\right)\left(\frac{z - z_1}{b_z}\right)$$

$$h(x_2, y_2, z_1) = h(x_2, y_2, z_1) + w\left(\frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right)\left(1 - \frac{z - z_1}{b_z}\right)$$

$$h(x_2, y_2, z_2) = h(x_2, y_2, z_2) + w\left(\frac{x - x_1}{b_x}\right)\left(\frac{y - y_1}{b_y}\right)\left(\frac{z - z_1}{b_z}\right)$$

2. If using R2-HoG, follow step 1 individually for both first and second order image gradients, i.e. create two 3-D histograms.
3. If using C-HoG,
   a. Divide the image region centered on the point into a log-polar circular block; create angular and radial bins dividing the block into cells
   b. Create a β bin orientation histogram for each cell

For each pixel in the block, vote into the cell histograms using trilinear interpolation in log-polar orientation space.

Algorithm 2.4. *HoG and its vaiants, Descriptor Computations*

Common Final Steps:

1. Apply *L2-Hys* or *L1-sqrt* normalization independently to each block; if using R2-HoG, apply normalization to each 3-D histogram independently.

$$\text{L2} - \text{Hys: } \begin{cases} v = \dfrac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}, & \|v\| < 0.2 \\ v = 0.2, & \|v\| > 0.2 \end{cases}$$

2. Collect the HoG for all the blocks lying in the image window into one big descriptor vector.

Some considerations should be made for the parameters in HoG which are summarized below:

1. Overlap between adjacent block in the image: in direct proportion with classification rate, *50* % is an standard value
2. Scale (σ): in reverse proportion with classification rate, *0* is the best value, relatively coarse spatial binning.
3. Number of orientation bins (β): in direct proportion with classification rate, *9* is a standard value (in the range of *0* to *180* degrees and unsigned gradient orientations), fine orientation binning.
4. Normalization methods: Best classification rate has been obtained by *L2-Hys*, *L2-Norm*, *L1-sqrt*, high quality local contrast normalization.

Algorithm 2.4. *HoG, Common Final Steps*

It is worth to mention that HoG is slower and more computationally intensive than integral images of Viola, Jones 2001. A simple visualization of HoG feature set for a human image is given in figure 2.13.

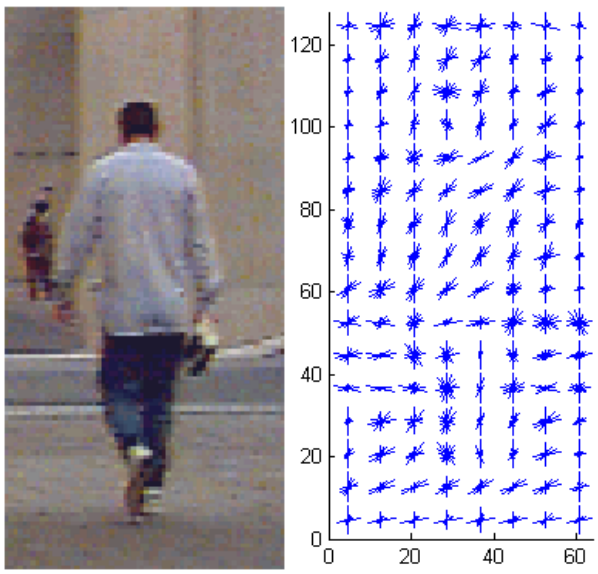


Figure 2.13: *Visualization of HoG feature set for a human image*

2.1.9 Covariance

The basic idea for introducing the covariance of a set of features is to get a more robust feature set towards different types of image and object changes. In this section, we first bring up with some mathematics behind the covariance method. Then, we show some possible applications of this method in the field of visual object tracking.

Given a feature vector for each pixel of an image/target, the goal is to compute the covariance matrix corresponding to the whole area of the image/target. Finally, we would be able to represent the image/target by this covariance matrix. In other words, this covariance matrix could be considered as an alternative to previous set of features, as later will be observed in this section.

For simplicity, assume that the feature vector is computed simply by getting the following features from each pixel of an image:

$$\mathbf{f}_i \equiv \mathbf{f}(x,y) = [x, y, I(x,y), |I_x|, |I_y|, |I_{xx}|, |I_{yy}|, \sqrt{\left(|I_{xx}|^2 + |I_{yy}|^2 \right)}, tan^{-1}\left(\frac{I_y}{I_x}\right)] \quad (2.22)$$

where I_x, I_{xx} are the 1st and 2nd order gradients in x direction respectively.

Afterwards, the feature matrix **F** is formed by all the feature vectors as below ($n=9$):

$$\mathbf{F} = \begin{bmatrix} \mathbf{f}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{f}_N \end{bmatrix}, \mathbf{F}: N * n \quad (2.23)$$

where N is the number of image points from each of which n features (in this case, 9) is extracted. The so called Covariance Matrix of this feature matrix is built upon the following equation:

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^{n} (\mathbf{F} - \boldsymbol{\mu})^T (\mathbf{F} - \boldsymbol{\mu}), \mathbf{C}: n \times n \quad (2.24)$$

where $\boldsymbol{\mu}$ is a matrix with all its rows equal to the average row vector of all the N rows in **F**. As the number of features for $\mathbf{f}_i$ is 9 in this case, we get 9×9 covariance matrix for each candidate region, irrespective of the dimensions for the image/target or the number of image points considered.

As the formula implies, **C** contains the statistical properties for the features of all the pixels in the image/target. This gives us a great advantage over the feature vectors

themselves. Here, we don't give a mathematical proof for this claim. However, intuitively it is quite clear that by having the interrelation of these features belonging to different pixels in addition to having the features for each individual pixel, we get much more robustness towards different types of image/target changes. In other words, if image/target goes under some changes, the covariance matrix of image/target before and after change would be quite similar.

A possible application of this method in the field of visual object tracking is the situation in which a predetermined region of interest is to be tracked with a certain speed and without drastic appearance changes. Then, in the consecutive frames, one may define different candidate regions around previous area of the target. Then the covariance matrix for all these candidate regions is calculated. Afterwards, a measure of similarity is defined based on the covariance matrices which is:

$$Similarity = \sqrt[2]{\sum_i (\log(v_i))^2} \tag{2.25}$$

in which v_i is the generalized eigenvalues (λ's) for a solution to the following equation:

$$\mathbf{C}_1 x = \lambda \mathbf{C}_2 \mathbf{x} \tag{2.26}$$

In this equation, $\mathbf{C}_1$ and $\mathbf{C}_2$ are the 2 covariance matrices to be compared and assigned a similarity measure and **x** is the corresponding generalized eigenvectors. As λ becomes smaller, the similarity between $\mathbf{C}_1$ and $\mathbf{C}_2$ is considered to be higher.

2.1.10 PCA

If the dimensions of the covariance matrix in previous section is large (i.e. the number of features), the calculation of eigenvalues for such a matrix is computationally intensive. In order to reduce the dimensionality of this matrix, one idea is to use Principal Component Analysis or PCA to extract only the most prominent information from the covariance matrix.

So let's say that we have a 32×32 image region containing an object to be described by the covariance method. In this section, we treat the covariance method in a different way. First, we derive the mean object by averaging this region through for example 15 frames. Then, the mean object is subtracted from each image and then the covariance matrix is calculated over the whole 15 frames. Hence, the resulting covariance matrix holds the relation of each image to itself and other images as well.

Denoting 2D images $\mathbf{A}_i$ ($r\times c$) by column-scanned vectors $\mathbf{a}_i$ ($rc\times 1$), the sample covariance matrix (based on the mean image, not features matrix) can be computed as

$$\mathbf{C} = \frac{1}{l}\sum_{i=1}^{l}(\mathbf{a}_i - \bar{\mathbf{a}})(\mathbf{a}_i - \bar{\mathbf{a}})^T = \frac{1}{l}\mathbf{D}\mathbf{D}^T \tag{2.27}$$

Now that we have the covariance matrix ($rc\times rc = 1024\times 1024$), we are looking for eigenvectors and eigenvalues of this matrix to reduce the dimensionality of data. In [30], it has been suggested to break this covariance matrix into two parts which are row projected and column projected covariance matrices assuming that the transform kernels along rows and columns in an image region are separable. This method is named as 2DPCA. In this case, we have two covariance matrices to calculate eigenvalues and eigenvectors for which are of size ($r\times r$) and ($c\times c$) which is (32×32) in this case. This calculation is much more efficient than previous one. In other words, the covariance matrix is calculated once over the rows and then over the columns following by the calculation for corresponding eigenvectors and eigenvalues in each case. The outer product between each pair of row and column-eigenvectors corresponds to what is called a basis image. The row-projected covariance matrix, for instance, holds the relation between the rows of each image to its rows and the rows of other images in the image sequence.

In each of the methods mentioned above, the largest eigenvalues and corresponding eigenvectors are chosen. Let's choose 4 row-projected and 4 column projected eigenvectors and eigenvalues to form 16 basis images by outer product of

their different pairs. This means that for 15 images, 16 basis images represent the statistical features of an object of interest in these images. Later, these eigenvectors and eigenvalues may be used for reconstruction of object in each image by some error. Such image (specifically for a human face) is also known as eigenface in literature [28,29].

In figure 2.14, 2.15, 2.16 the concept of PCA is visualized through a block diagram and some results on 10 face images.

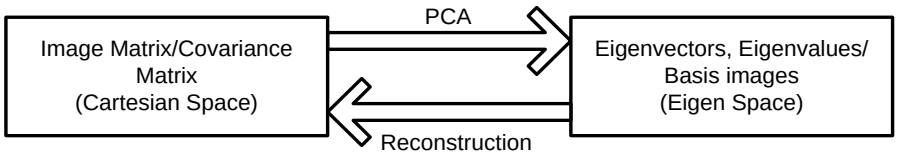


Figure 2.14: *Block diagram of PCA*

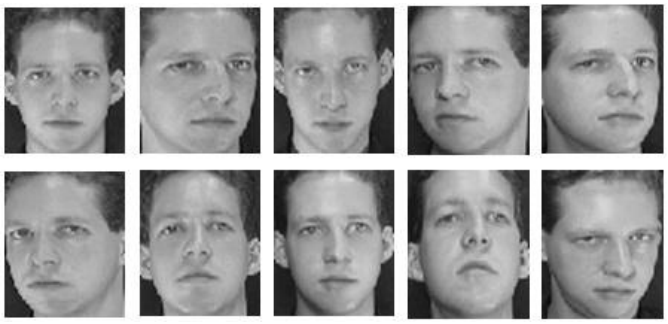


Figure 2.15: *10 images of a human face in a video*

Figure 2.16: *10 basis images of a human face from figure 2.15*

In algorithm 2.5, 2.6 the steps for PCA and 2DPCA are summarized.

1. Get l images $\mathbf{A}_i(r \times c)$ from a sequence.

2. For each $\mathbf{A}_i$, calculate the corresponding column format mean subtracted image d_i:

$$\mathbf{A}_i = \begin{bmatrix} a_{1,1}^i & \cdots & a_{1,c}^i \\ \vdots & \ddots & \vdots \\ a_{r,1}^i & \cdots & a_{r,c}^i \end{bmatrix}$$

$$\mathbf{a}_i = \begin{bmatrix} a_{1,1}^i \\ \vdots \\ a_{1,c}^i \\ \vdots \\ a_{r,1}^i \\ \vdots \\ a_{r,c}^i \end{bmatrix}, \bar{\mathbf{a}} = \begin{bmatrix} \frac{1}{l}\sum_{i=1}^{l} a_{1,1}^i \\ \vdots \\ \frac{1}{l}\sum_{i=1}^{l} a_{r,c}^i \end{bmatrix}, \mathbf{d}_i = \mathbf{a}_i - \bar{\mathbf{a}} = \begin{bmatrix} a_{1,1}^i - \frac{1}{l}\sum_{i=1}^{l} a_{1,1}^i \\ \vdots \\ a_{r,c}^i - \frac{1}{l}\sum_{i=1}^{l} a_{r,c}^i \end{bmatrix}$$

$$\mathbf{D}_{l \times rc} = [\mathbf{d}_1\ \mathbf{d}_2\ \ldots\ \mathbf{d}_l]^{\mathrm{T}}$$

3. Derive the covariance matrix of column-scanned mean subtracted images **D**:

$$\mathbf{C}_{l \times l} = \frac{1}{l}\mathbf{D}\mathbf{D}^{\mathrm{T}}$$

4. Calculate eigenvectors and eigenvalues of the covariance matrix.

$$\mathbf{C}_{l \times l}\mathbf{V}_{l \times l} = \mathbf{V}_{l \times l}\mathbf{E}_{l \times l}$$

V: eigenvectors as columns, **E**: eigenvalues on the diagonal in ascending order

5. Sort eigenvectors based on corresponding eigenvalues in descending order.

6. Change these eigenvectors of $\mathbf{D}\mathbf{D}^{\mathrm{T}}$ to $\mathbf{D}^{\mathrm{T}}\mathbf{D}$ by the following math trick:

$$\mathbf{D}\mathbf{D}^{\mathrm{T}}\mathbf{V} = \mathbf{V}\mathbf{E}$$

$$\times \mathbf{D}^{\mathrm{T}} \rightarrow \mathbf{D}^{\mathrm{T}}\mathbf{D}(\mathbf{D}^{\mathrm{T}}\mathbf{V}) = (\mathbf{D}^{\mathrm{T}}\mathbf{V})\mathbf{E}$$

$$\rightarrow \mathbf{C}_{\mathrm{new}} = \mathbf{D}^{\mathrm{T}}\mathbf{D}, \mathbf{V}_{\mathrm{new}} = \mathbf{D}^{\mathrm{T}}\mathbf{V}$$

So, now the eigenvalues **E** and eigenvectors $\mathbf{V}_{\mathrm{new}}$ correspond to the real covariance matrix of images that is $\mathbf{D}^{\mathrm{T}}\mathbf{D}$. It is worth to note that these eigenvectors are orthonormal and they represent the relation of each image $\mathbf{a}_i$ to other images in the sequence and $\mathbf{a}_i$ itself. (Simple proof of orthonormal by having **C** as symmetric positive matrix and definition of **C**)

7. Pick up optimal d number of eigenvectors corresponding to the first d largest eigenvalues. This optimal value for d could be easily defined as the total number of eigenvalues which reside between 65%-90% of total eigenvalues cumulative sum. These eigenvectors represent the basis images or eigenfaces (figure 2.16).

8. Calculate the projection of image $\mathbf{a}_i$ in eigenspace by:

$$\mathrm{projection}_{l \times 1} = (\mathbf{V}_{\mathrm{new}})^{\mathrm{T}} \times \mathbf{d}_i = \begin{bmatrix} v_{1,1} & \cdots & v_{1,rc} \\ \vdots & \ddots & \vdots \\ v_{l,1} & \cdots & v_{l,rc} \end{bmatrix} \begin{bmatrix} d_{1,1}^i \\ \vdots \\ d_{rc,1}^i \end{bmatrix}$$

9. For reconstructing the image $\mathbf{a}_i$, first reconstruct the mean subtracted image $\mathbf{d}_i$ by calculating the multiplication of transformed image from step 8 with orthonormal eigenvectors set to project it back to image space. Then calculate the summation of this reconstruction and the average image: (Figure 2.18)

$$\mathrm{reconstruction} = (\mathbf{V}_{\mathrm{new}}) \times \mathrm{projection}$$

$$\mathbf{a}_i = \mathrm{reconstruction} + \bar{\mathbf{a}}$$

Algorithm 2.5. *Conventional PCA*

1. Get l images $\mathbf{A}_i(r \times c)$ from a sequence.

2. Derive the row-projected and column-projected covariance matrices of Images.

$$\mathbf{F}_{r\times r} = \frac{1}{l}\sum_{i=1}^{l} \mathbf{B}_i \mathbf{B}_i{}^{T}$$

$$\mathbf{G}_{c\times c} = \frac{1}{l}\sum_{i=1}^{l} \mathbf{B}_i{}^{T} \mathbf{B}_i$$

$$\mathbf{B}_i = \mathbf{A}_i - \overline{\mathbf{A}}, \overline{\mathbf{A}} = \frac{1}{l}\sum_{i=1}^{l} \mathbf{A}_i$$

$\mathbf{A}_i$: 2D images $(r \times c)$, l: Number of images in a sequence

3. Calculate eigenvectors and eigenvalues of the two covariance matrices.

$$\mathbf{F}_{r\times r}\mathbf{V}^{F}{}_{r\times r} = \mathbf{V}^{F}{}_{r\times r}\mathbf{E}^{F}{}_{r\times r}$$

$$\mathbf{G}_{c\times c}\mathbf{V}^{G}{}_{c\times c} = \mathbf{V}^{G}{}_{c\times c}\mathbf{E}^{G}{}_{c\times c}$$

$\mathbf{V}$: eigenvectors as columns, $\mathbf{E}$: eigenvalues on the diagonal in ascending order

4. Sort eigenvectors based on corresponding eigenvalues in descending order.

5. Pick up optimal d number of eigenvectors corresponding to the first d largest eigenvalues. This optimal value for d could be found the same as with Conventional PCA.

6. Calculate the projected or transformed mean subtracted image or Principal Components of mean subtracted image $\mathbf{A}_i - \overline{\mathbf{A}}$:

$$\text{projection} = (\mathbf{V}^{F})^{T} \times (\mathbf{A}_i - \overline{\mathbf{A}}) \times \mathbf{V}^{G}$$

7. For reconstructing the image from its row and column principal components, first project back the transformed image from eigen space to image space and then calculate its summation with the average image: (Figure 2.19)

$$\text{reconstruction} = \mathbf{V}^{F} \times \text{projection} \times \left(\mathbf{V}^{G}\right)^{T}$$

$$\mathbf{A}_i = \text{reconstruction} + \overline{\mathbf{A}}$$

Algorithm 2.6. *(2D)[2] PCA*

Table 2.3: *Comparison Between Different Versions of PCA*

Version	Covariance Matrix Size (Input)	Eigenvector Size (Output)	Computation(P^2)	Computation(R^3)
PCA	*rc×rc or l×l*	*r×c or l*	More Intensive	More Efficient
2DPCA	*c×c*	*c*	Less Intensive	Less Efficient
E2DPCA	*r×r,c×c*	*r+c*	Less Intensive	Least Efficient

l: Number of images in a video sequence or face database;
r: Number of rows for each image or frame;
c: Number of columns for each image or frame

[2] Projection
[3] Reconstruction

The reconstruction of an image by its basis images is simplified in the following block diagram:

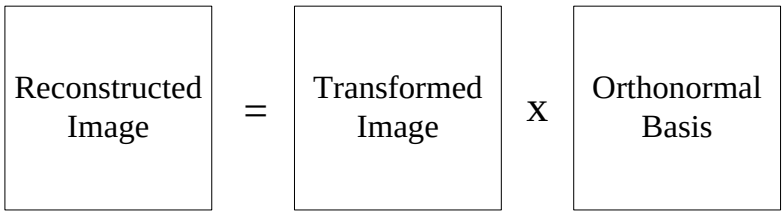


Figure 2.17: *the block diagram for reconstruction of an image from its basis images*

In the following figures, the result of reconstructing the so called eigenfaces with a portion of their eigenvectors and eigenvalues are shown.

Figure 2.18: *Reconstructed Face images by PCA using (1 to 10) eigenvectors (From left to right, up to down)*

Figure 2.19: *Reconstructed Face images by 2DPCA using (1 to 20) eigenvectors*

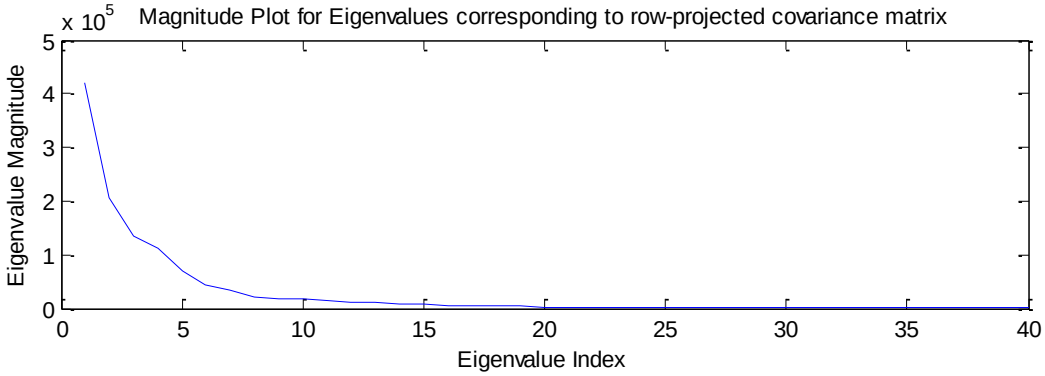


Figure 2.20: *Magnitude plot for eigenvalues of row-projected covariance matrix*

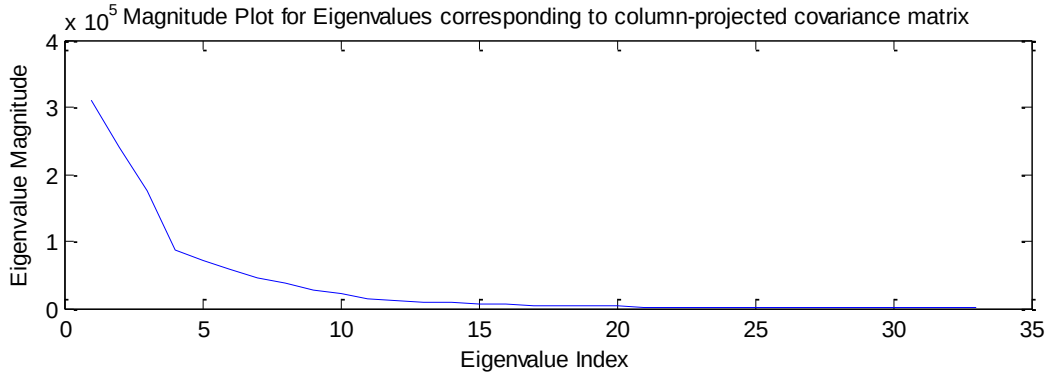


Figure 2.21: *Magnitude plot for eigenvalues of column-projected covariance matrix*

2.2 Visual Object Tracking Using Multiple Cameras

In this part, some topics from multiple-view geometry are discussed with their application to this thesis. These topics are useful for better understanding the three-dimensional structure of a scene by having its multiple views. Besides, one may be able to derive the corresponding camera configuration within this framework.

In order to make a relationship between multiple images of an object in multiple views, normally it's assumed that the intrinsic and extrinsic camera parameters are known beforehand. However, in this thesis, the assumption is having dynamic cameras (i.e. neither intrinsic nor extrinsic camera parameters are known). Therefore, as described later in this section, a virtual ground plane is defined up to some projective ambiguity in 3D space on which we may be able to make such relationship between multiple images of an object.

The aim in this part is to explain the geometric and algebraic constraints that hold among two and three views of the same scene. First it is shown that the first image of any point in the 3D world coordinate system must lie in the plane formed by its second image and the optical centres of the corresponding cameras. This constraint is a part of so called epipolar geometry which will be algebraically described under the name of fundamental matrix [1].

2.2.1 Camera Parameters

In this section, the camera matrix is factorized and explained by its so called intrinsic and extrinsic camera parameters. Such decomposition gives a clear view of the camera model and how it maps the coordinates of the object from 3D world coordinate system to the (camera/image plane/pixel) coordinate system. In figure 2.22, a hierarchy of steps for such conversion is shown. More importantly, the camera model that is considered here obeys the pin-hole camera model. The interested reader is recommended to read in literature about these kinds of image formation [2].

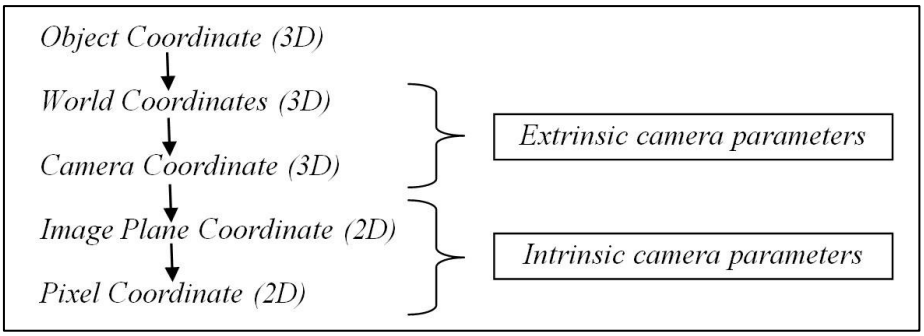


Figure 2.22: *Hierarchy of Coordinate Conversion steps in a camera model*

❖ **Note**: Object coordinates will be described later in section 2.2.1.1.

In general, the world and pixel coordinate systems are related by a set of physical parameters such as:

- ✓ the focal length of the lens (*f*)
- ✓ the size of the pixels (for example, measured in meters or inches)
- ✓ the position of the principal point *(u0,v0)* (intersection of optical axis with the image plane)
- ✓ the position and orientation of the camera (with respect to a fixed 3D world coordinate system)

These set of physical parameters are visualized in figure 2.23 for more clarity.

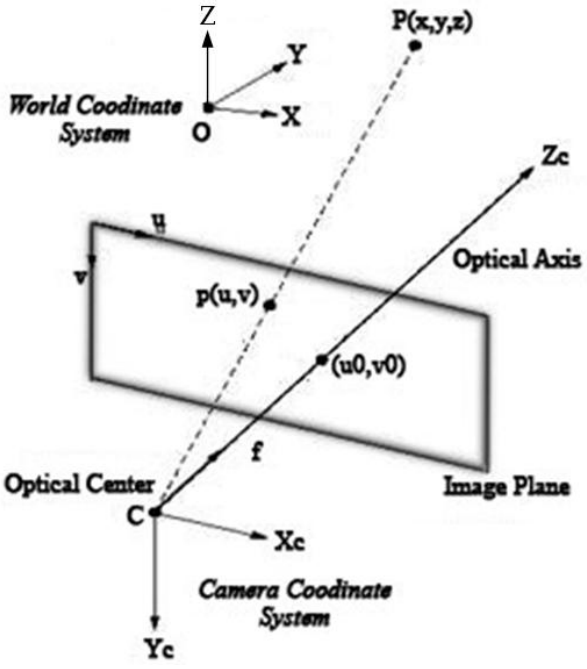


Figure 2.23: *Visualization of physical Camera parameters*

Two types of parameters mentioned in figure 2.23 need to be recovered in order to reconstruct the 3D structure of a scene from the pixel coordinates of its image points or vice versa. Specifically, these set of parameters can be categorized into two general types:

- ✓ *Extrinsic camera parameters:* the parameters that define the location and orientation of the camera reference frame with respect to a known world reference frame.
- ✓ *Intrinsic camera parameters:* the parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

In other words, the extrinsic camera parameters uniquely identify the transformation between the unknown camera reference frame and the known world reference frame. On the other hand, the intrinsic camera parameters characterize the optical, geometric, and digital characteristics of the camera:

(1) The perspective projection (focal length *f*)

(2) The transformation between image plane coordinates and pixel coordinates

(3) The geometric distortion introduced by the optical lens

Right now, we are able to derive the relationship among different coordinate systems with the help of these parameters. According to the steps in figure (2.22), each step is described in a separate section below. Observe figure 2.24 for a better understanding of this section.

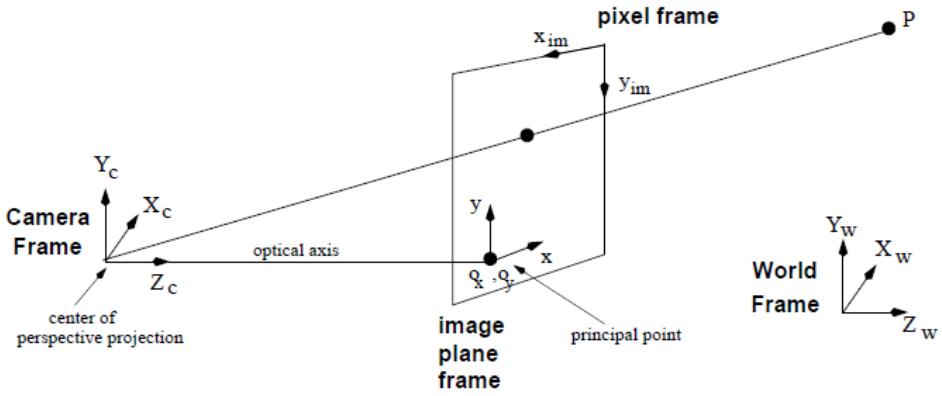


Figure 2.24: *Different Coordinate Systems and Camera Parameters*

2.2.1.1 From 3D Object to World Plane Coordinates:

For conversion between object and world plane coordinates, one needs to calibrate the scene, i.e. setting some predefined equalities between the unit of measurement in these two frameworks. As shown in figure 2.25, 1 unit in either x or y direction on the world plane is equal to 1.2 meters in the real 3D world coordinates. Therefore, the object coordinates in meters are explicitly converted to some world plane coordinates by arbitrarily setting up two basic assumptions:

- ✓ Consider the origin of world coordinate system fixed and arbitrarily chosen in the scene.
- ✓ Set some equalities between the unit of measurements in these 2 frameworks, such as:

$$\mathbf{X}_O = \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix} \Rightarrow \mathbf{X} = \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix} = \begin{bmatrix} \alpha x_O \\ \beta y_O \\ \gamma z_O \end{bmatrix} \tag{2.28}$$

Where in the case of figure 2.25, we would have the following parameter values:

$$\begin{cases} \alpha = 1/1.2(meters) \\ \beta = 1/1.2(meters) \\ \gamma = 1/1.2(meters) \end{cases}$$

Figure 2.25: *Calibrated Scene for PETS 2007*

Table 2.4: *Object to World Coordinates Conversion for figure 2.25*

Point#	Coordinates					
	Object Coordinates (meters)			World Coordinates (optional units)		
	x_O	y_O	z_O	x_W	y_W	z_W
1	1.2	1.2	0	1	1	0
2	2.4	-1.2	0	2	-1	0

2.2.1.2 From World Plane to Camera Coordinates:

Using the extrinsic camera parameters, the relation between the coordinates of a point in the 3D world (**X**) and camera (**x**) coordinates will become:

$$\mathbf{x} = \mathbf{R}(\mathbf{X} - \mathbf{t}) \tag{2.29}$$

where

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.30}$$

If we rewrite **x** and **X** as vectors, we get:

$$\mathbf{x} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \Rightarrow \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w - t_x \\ y_w - t_y \\ z_w - t_z \end{bmatrix} \tag{2.31}$$

2.2.1.3 From Camera to Image Plane Coordinates:

For transferring the coordinates from camera reference frame to image reference frame, a perspective projection is applied to the result of previous section, i.e. **x**:

$$x = f\frac{x_c}{z_c} = f\frac{\mathbf{R}_1^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})}, \quad y = f\frac{y_c}{z_c} = f\frac{\mathbf{R}_2^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} \tag{2.32}$$

Where

X: The coordinates of a point in 3D world.

R: The rotation matrix derived by the camera extrinsic parameters.

$\mathbf{R}_i^T$: The i[th] row of $\mathbf{R}_i^T$.

T: The translation vector derived by the camera extrinsic parameters.

2.2.1.4 From Image Plane to Pixel Coordinates:

Since the result of previous step is with respect to the focal point of the camera and they are not expressed in units of pixels, they need to be corrected by applying the following scaling and translation transformations corresponding to camera intrinsic parameters:

$$\begin{cases} x = -(x_{im} - o_x)s_x \quad or \quad x_{im} = {}^{-x}/_{s_x} + o_x \\ y = -(y_{im} - o_y)s_y \quad or \quad y_{im} = {}^{-y}/_{s_y} + o_y \end{cases} \tag{2.33}$$

where (o_x , o_y) are the coordinates of the principal point (in pixels, e.g., $o_x = N/2$, $o_y = M/2$ if the principal point is the centre of the image. *N* and *M* are the width and height of the camera image respectively) and s_x , s_y correspond to the effective size of the pixels in the horizontal and vertical directions (e.g. in millimetres). By using matrix notation:

$$\begin{bmatrix} x_{im} \\ y_{im} \\ 1 \end{bmatrix} = \begin{bmatrix} -1/s_x & 0 & o_x \\ 0 & -1/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.34}$$

The five intrinsic parameters f, s_x, s_y, o_x, o_y are not independent. But we can define the following four independent parameters:

$f_x = f/s_x$: the focal length in horizontal pixels.
$\alpha = s_y/s_x$ ($\alpha = f_x/f_y$): aspect ratio.
o_x, o_y: image centre coordinates.

So, by joining the results from all the previous steps, we come to a single equation that relates pixel coordinates to world coordinates:

$$-(x_{im} - o_x)s_x = f\frac{\mathbf{R}_1^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})}, \quad -(y_{im} - o_y)s_y = f\frac{\mathbf{R}_2^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})}$$

or

$$x_{im} = -fs_x\frac{\mathbf{R}_1^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} + o_x, \quad y_{im} = -fs_y\frac{\mathbf{R}_2^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} + o_y \tag{2.35}$$

Last, in case of radial lens distortion, two more intrinsic parameters are introduced to compensate for this error, as in the following two equations:

$$\begin{cases} x = x_d(1 + k_1r^2 + k_2r^4) \\ y = y_d(1 + k_1r^2 + k_2r^4) \end{cases} \tag{2.36}$$

where (x_d, y_d) are the coordinates of the distorted points ($r^2 = {x^2}_d + {y^2}_d$) and k_1 and k_2 are intrinsic parameters too, but for the sake of simplicity, they will not be considered here.

Since in this thesis, the main focus is on the uncalibrated cameras, we don't enter into details for estimation of these camera parameters directly. However, we may estimate the whole camera matrix by using Epipolar Geometry, as described in section 3.2.

2.2.2 3D Transformation Models

In this section, an extension of 2D transformation models will be presented. However, the basic idea is remained the same as in section 2.1.5. The most general transformation model is the affine transformation, where changes in position, size and shape are allowed. The 3D affine transformation is widely used in computer vision and particularly, in the area of model based object recognition. The principal

parameters involved can be from 8 to 12, the number depends of the kind of relationship exploited:

- 12 parameters affine transformation are used to define relationship between two 3D images;
- 9 parameters can be used in reconstructing.
- 8 parameters to describe a model that transform 3D object space to 2D image space.

An affine transformation includes at the same time three basic types of transformations: translation, rotation and scaling. In matrix format and in homogeneous coordinate system, one can write:

I. Translation: $\mathbf{X}_j = \mathbf{T}(dx, dy, dz)\mathbf{X}_i$

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.37}$$

II. Scale: $\mathbf{X}_j = \mathbf{S}\big(s_x, s_y, s_z\big)\mathbf{X}_i$

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.38}$$

III. Rotation: $\mathbf{X}_j = \mathbf{R}_z(\theta)\mathbf{X}_i$ (with respect to z-axis)

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.39}$$

$$x_j = x_i\, cos(\theta) - y_i\, sin(\theta)$$
$$y_j = x_i\, sin(\theta) + y_i\, cos(\theta)$$
$$z_j = z_i$$

2.2.3 Epipolar Geometry

In this section, a clear basis is made for quantifying the relationship between multiple cameras, the object of interest in the 3D world coordinate system and its images in the camera image planes by epipolar geometry. It is assumed that the scene points are visible in all the camera image planes.

Within the subject of multiple view geometry, there is a relationship between two images of a scene from two cameras or from a moving camera. In particular, given a point **X** in the scene (i.e. 3D world coordinate system), there exists a relationship between its projections in multiple camera image planes. Epipolar geometry describes all these relationships. Without considering the scene structure, one may derive a correspondence between projection of scene points in multiple views without knowing the intrinsic and extrinsic camera parameters.

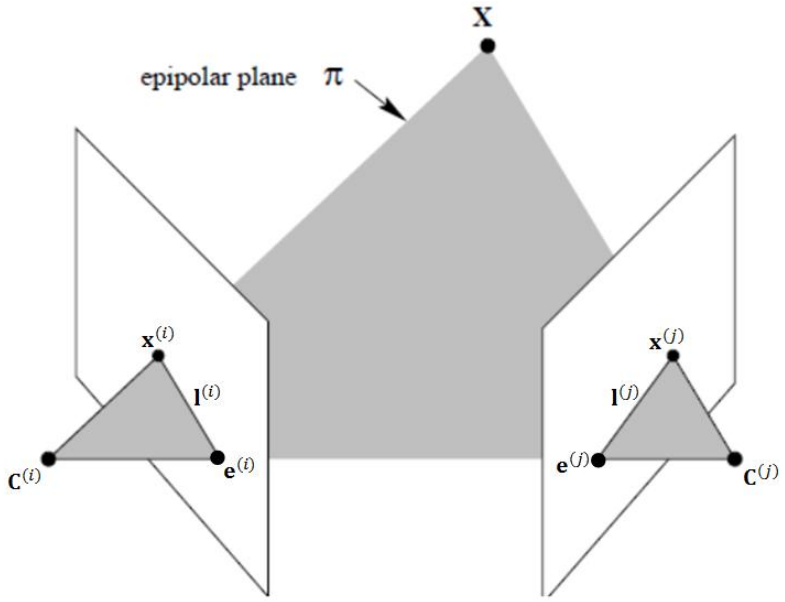


Figure 2.26: *Relation between 2 images a given point X in 3D world coordinate system by epipolar geometry. This figure is partly taken from [1].*

As we know from linear algebra, a line which is represented by $ax + by + cz - d = 0$ can be rewritten in orthonormal coordinate frame as:

$$(a, b, c, -d)\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \tag{2.40}$$

or more concisely, as

$$\prod \cdot \mathbf{X} = 0 \tag{2.41}$$

The vector **X** is called the vector of homogenous coordinates corresponding to point **X** in the 3D world coordinate system. See appendix A for more details about homogeneous coordinate system. As shown in figure 2.26, the two cameras have their centres at $C^{(i)}$, $C^{(j)}$. Within this configuration, epipolar geometry expresses a mapping between a point $\mathbf{x}^{(i)}$ in one image plane and a line $\mathbf{l}^{(j)}$ (named as epipolar line) in the other camera image plane on which the other point $\mathbf{x}^{(j)}$ lies. This mapping is also known as Epipolar Constraint in literature [1].

The epipolar constraint is the key to make the correspondence between points in different views without calibration. The main benefit is that the search space for the corresponding point $\mathbf{x}^{(j)}$ is limited to the epipolar line $\mathbf{l}^{(j)}$, see figure 2.27, i.e. the search space is reduced from 2 dimensions to 1.

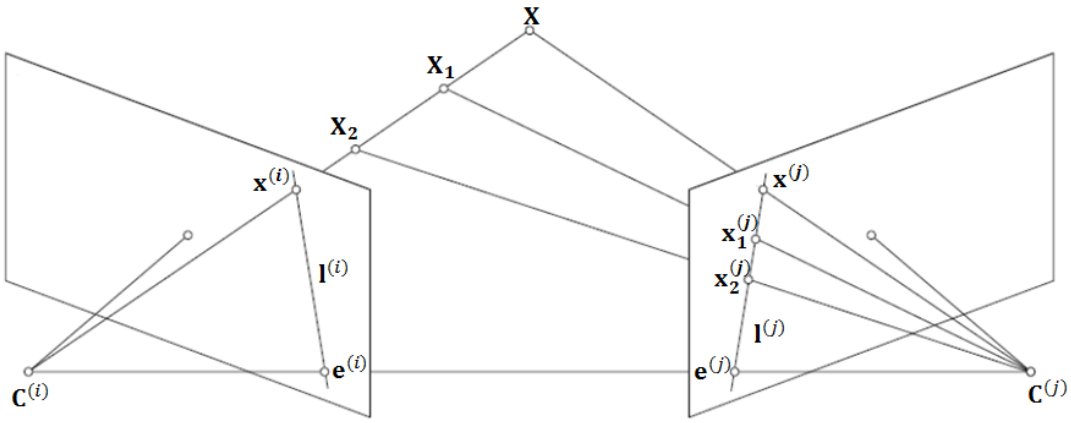


Figure 2.27: *Epipolar constraint-the set of candidate points corresponding to* $\boldsymbol{x}^{(i)}$ *is constrained to lie on the associated epipolar line* $\boldsymbol{l}^{(j)}$*. This figure is partly taken from [1].*

The epipolar constraint can be algebraically represented by *a 3×3* matrix which is known as fundamental matrix **F**. If the camera intrinsic parameters are assumed to be known, this matrix is also called as essential matrix. In fact, the fundamental matrix is a generalized form for essential matrix in the case we don't have the calibration matrix for the camera (i.e. camera intrinsic parameters *K*) [5].

In this thesis, only the fundamental matrix is considered. In the next section, we come up with the basic technique for estimating the fundamental matrix **F** from two uncalibrated images, without having the camera intrinsic and extrinsic parameters. For

more clarity, the basic elements of epipolar geometry are defined in the list below. Also look again at figure 2.26.

- ***epipole*:** the point of intersection of the line joining the camera centers with the image plane. Equivalently, the epipole is the image in one view of the camera center of the other view.
- ***epipolar plane:*** a plane containing **X** and the baseline (a line connecting the two camera centers which also passes through epipoles).
- ***epipolar line:*** the intersection of an epipolar plane with the image planes. All the epipolar lines intersect at the epipole.

2.2.4 Fundamental Matrix

In this part, an algebraic representation of epipolar constraint is given following by a description for the fundamental matrix **F**. Later on, some interesting properties of **F** are observed. Epipolar Constraint can be algebraically represented in the following form:

$$(\mathbf{x}^{(j)})^T \mathbf{F}_{ij}\, \mathbf{x}^{(i)} = 0 \tag{2.42}$$

This equation simply says that the point in the second image $\mathbf{x}^{(j)}$ is collinear with the correspondent epipolar line $\mathbf{l}^{(j)}$ that is derived by $\mathbf{F}_{ij}$ (i.e. $\mathbf{F}_{ij}\, \mathbf{x}^{(i)}$). For more clarity, look at figure 2.28.

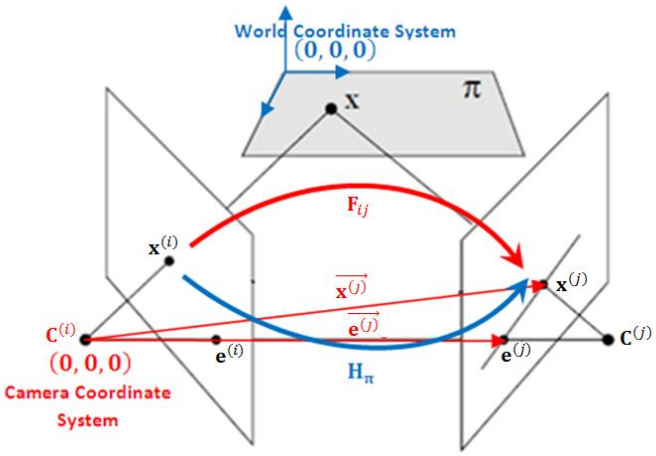


Figure 2.28: *Epipolar Constraint, Fundamental Matrix and Homography. This figure is partly taken from [1].*

The plane π in figure 2.28 is determined by the baseline and the ray defined by $\mathbf{x}^{(i)}$. The ray corresponding to the other point $\mathbf{x}^{(j)}$ lies in π, hence the point $\mathbf{x}^{(j)}$ lies

on the line of intersection ($\mathbf{l}^{(j)}$) of π with the second image plane. In other words, a point $\mathbf{x}^{(i)}$ in one image is transferred via the plane π to its corresponding point $\mathbf{x}^{(j)}$ in the other image. The epipolar line $\mathbf{l}^{(j)}$ is formed by joining $\mathbf{x}^{(j)}$ with the epipole $\mathbf{e}^{(j)}$.

By having such geometric configuration and from what we know about epipolar constraint, the epipolar line can be algebraically expressed as $\mathbf{l}^{(j)} = \mathbf{e}^{(j)} \times \mathbf{x}^{(j)}$ and can be rewritten as follows:

$$\mathbf{l}^{(j)} = \left[\mathbf{e}^{(j)}\right]_x \mathbf{x}^{(j)} \tag{2.43}$$

where

$$\mathbf{e}^{(j)} = \begin{bmatrix} e_1^{(j)} \\ e_2^{(j)} \\ e_3^{(j)} \end{bmatrix}, \mathbf{x}^{(j)} = \begin{bmatrix} x_1^{(j)} \\ x_2^{(j)} \\ x_3^{(j)} \end{bmatrix}, \quad \left[\mathbf{e}^{(j)}\right]_x = \begin{bmatrix} 0 & -e_3^{(j)} & e_2^{(j)} \\ e_3^{(j)} & 0 & -e_1^{(j)} \\ -e_2^{(j)} & e_1^{(j)} & 0 \end{bmatrix} \tag{2.44}$$

or in the expanded form:

$$\mathbf{l}^{(j)} = \mathbf{e}^{(j)} \times \mathbf{x}^{(j)} = \begin{vmatrix} i & j & k \\ e_1^{(j)} & e_2^{(j)} & e_3^{(j)} \\ x_1^{(j)} & x_2^{(j)} & x_3^{(j)} \end{vmatrix} = \begin{bmatrix} 0 & -e_3^{(j)} & e_2^{(j)} \\ e_3^{(j)} & 0 & -e_1^{(j)} \\ -e_2^{(j)} & e_1^{(j)} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(j)} \\ x_2^{(j)} \\ x_3^{(j)} \end{bmatrix} \tag{2.45}$$

By introducing the homography matrix $\mathbf{H}_\pi$ mapping each $\mathbf{x}^{(i)}$ to $\mathbf{x}^{(j)}$, we can rewrite equation (2.45) in the following form:

$$\mathbf{l}^{(j)} = \left[\mathbf{e}^{(j)}\right]_x \mathbf{H}_\pi \mathbf{x}^{(i)} = \mathbf{F}_{ij}\, \mathbf{x}^{(i)} \tag{2.46}$$

where $\mathbf{F}_{ij}$ is the Fundamental Matrix and $\mathbf{H}_\pi$ is the 2D planar homography which is later described more thoroughly in section 2.2.7. The notation $[\mathbf{a}]_x$ is a skew-symmetric *3x3* matrix corresponding the vector **a**. A skew-symmetric matrix is singular and is its null-vector. Besides, the cross product of two 3-vector $\mathbf{a} \times \mathbf{b}$ is algebraically represented as $(a_2b_3 - a_3b_2\,, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)^T$ which can be rewritten using skew-symmetric matrix notation as:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_x \mathbf{b} = (\mathbf{a}^T [\mathbf{b}]_x)^T \tag{2.47}$$

where

$$[\mathbf{a}]_x = \begin{bmatrix} 0 & -a_3 & -a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{2.48}$$

It is evident from equation 2.42 that $\mathbf{F}_{ij}$ should be of rank 2. Besides, $\mathbf{F}_{ij}$ is only defined up to a scalar factor, because if $\mathbf{F}_{ij}$ is multiplied by an arbitrary scalar, equation (2.42) still holds. As a result, there are only 8 independent parameters among the 9 elements of the fundamental matrix $\mathbf{F}_{ij}$ which should be determined. A short list of properties for the fundamental matrix $\mathbf{F}_{ij}$ is provided here:

***Point correspondence*:**

- if $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are corresponding image points, then equation 2.42 holds.

***Epipolar lines*:**

- $\mathbf{l}^{(j)} = \mathbf{F}_{ij}\mathbf{x}^{(i)}$ is the epipolar line in the j^{th} image corresponding to $\mathbf{x}^{(i)}$ in the i^{th} image.
- $\mathbf{l}^{(i)} = \mathbf{F}_{ij}^{T}\mathbf{x}^{(j)}$ is the epipolar line in the i^{th} image corresponding to $\mathbf{x}^{(j)}$ in the j^{th} image.

***Epipoles*:**

- $\mathbf{F}_{ij}\mathbf{e}^{(i)} = 0$
- $\mathbf{F}_{ij}^{T}\mathbf{e}^{(j)} = 0$

In this section, a basic numerical method will be described for estimating the fundamental matrix given a set of point correspondences between views. There are different methods in the literature [1]. In this thesis, the main focus will be on the simplest one which is named as normalized 8-point algorithm. This method performs quite well if and only if the input data are normalized, as mentioned in the part (i) of algorithm 2.7.

Given sufficient point matches $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ (at least 8 for normalized 8-point algorithm), the equation 2.42 can be used to compute the fundamental matrix $\mathbf{F}_{ij}$. In particular, by expressing $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in homogeneous coordinates, each point pair gives rise to one linear equation in the unknown entries of $\mathbf{F}_{ij}$. Given $\mathbf{x}^{(i)} = \left(x^{(i)}, y^{(i)}, 1\right)^{T}$ and $\mathbf{x}^{(j)} = \left(x^{(j)}, y^{(j)}, 1\right)^{T}$, the equation (2.42) can be rewritten as follows:

$$(\mathbf{x}^{(j)})^{T}\mathbf{F}_{ij}\mathbf{x}^{(i)} = \begin{bmatrix} x^{(j)} & y^{(j)} & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x^{(i)} \\ y^{(i)} \\ 1 \end{bmatrix} = 0 \qquad (2.49)$$

$$= x^{(j)}(f_{11}x^{(i)} + f_{12}y^{(i)} + f_{13}) + y^{(j)}(f_{21}x^{(i)} + f_{22}y^{(i)} + f_{23}) + (f_{31}x^{(i)} + f_{32}y^{(i)} + f_{33})$$
$$= f_{11}(x^{(j)}\,x^{(i)}) + f_{12}(x^{(j)}y^{(i)}) + f_{13}x^{(j)} + f_{21}(x^{(i)}\,y^{(j)}) + f_{22}(y^{(i)}\,y^{(j)}) + f_{23}y^{(j)} + f_{31}x^{(i)} + f_{32}y^{(i)} + f_{33} = 0$$

By rearranging the above equation in the unknown elements of $\mathbf{F}_{ij}$, we get:

$$[x^{(i)}\,x^{(j)} \quad x^{(j)}y^{(i)} \quad x^{(j)} \quad y^{(j)}x^{(i)} \quad y^{(j)}y^{(i)} \quad y^{(j)} \quad x^{(i)} \quad y^{(i)} \quad 1]\begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \mathbf{Af} = 0 \qquad (2.50)$$

Therefore, given a set of *n* point matches, we obtain a set of linear equation of the form:

$$\mathbf{Af} = \begin{bmatrix} x_1^{(j)}x_1^{(i)} & x_1^{(j)}y_1^{(i)} & x_1^{(j)} & y_1^{(j)}x_1^{(i)} & y_1^{(j)}y_1^{(i)} & y_1^{(j)} & x_1^{(i)} & y_1^{(i)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^{(j)}x_n^{(i)} & x_n^{(j)}y_n^{(i)} & x_n^{(j)} & y_n^{(j)}x_n^{(i)} & y_n^{(j)}y_n^{(i)} & y_n^{(j)} & x_n^{(i)} & y_n^{(i)} & 1 \end{bmatrix}\mathbf{f} = 0 \qquad (2.51)$$

If more than 8 point correspondences are available, then one may compute **f** and hence $\mathbf{F}_{ij}$ by solving the linear set of equations (2.51) with the normalized 8-point algorithm. However, it is possible to compute $\mathbf{F}_{ij}$ with only 7 point correspondences by enforcing the singularity constraint on $\mathbf{F}_{ij}$ which says *det(*$\mathbf{F}_{ij}$*)=0*. Now, our purpose is to briefly describe the steps of the normalized 8-point algorithm.

Normalization part of algorithm 2.7 consists of a translation and scaling of each image pixel so that the centroid of the point coordinates is located at origin and the *RMS* distance of the points from the origin is equal to $\sqrt{2}$. For more detailed discussion about the reason for this kind of normalization, refer to chapter 4 of [1]. In part (ii) of the algorithm 2.7, there are two steps:

Linear solution. A solution $\mathbf{F}_{ij}$ is obtained from the vector **f** corresponding to the smallest singular value of **A**, where **A** is defined in the equation (2.51).

Constraint enforcement. Replace $\mathbf{F}_{ij}$ by $\mathbf{F'}_{ij}$, where $\mathbf{F'}_{ij}$ is the closest singular matrix to $\mathbf{F}_{ij}$ under a Frobenius norm. This correction is done using singular value decomposition or SVD which is briefly described in section 2.2.5.

For more accuracy on the final $\mathbf{F}_{ij}$, input point pairs must match precisely in order to get accurate results for $\mathbf{F}_{ij}$. In order to compensate for false point correspondences, we will use RANSAC along with a few of its variants, principally based on distance metric calculation and error minimization, which collaborates with the normalized 8-point algorithm to improve the accuracy of the computed $\mathbf{F}_{ij}$. Right after computing $\mathbf{F}_{ij}$, the planar homography $\mathbf{H}_{\pi}$ (to be explained in section 2.2.7) is derived according to algorithm 2.9 satisfying $\mathbf{x}^{(i)} \overset{\mathbf{H}_{\pi}}{\Leftrightarrow} \mathbf{x}^{(j)}$.

Input: $n \geq 8$ point correspondences $\{\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}\}$ expressed in homogeneous coordinates.

Output: The 3x3 Fundamental matrix, **F**, such that $(\mathbf{x}^{(j)})^T \boldsymbol{F} \mathbf{x}^{(i)} = 0$

(i) **Normalization:** Transform the image coordinates according to $\hat{\mathbf{x}}^{(i)} = \mathbf{T}_i \mathbf{x}^{(i)}$ and $\hat{\mathbf{x}}^{(j)} = \mathbf{T}_j \mathbf{x}^{(j)}$ where $\mathbf{T}_i$ and $\mathbf{T}_j$ are normalizing transformations consisting of a translation and scaling.

(ii) Find the fundamental matrix $\hat{\mathbf{F}}'_{ij}$ corresponding to the matches $\hat{\mathbf{x}}^{(i)} \leftrightarrow \hat{\mathbf{x}}^{(j)}$ by:

a) **Linear solution:** Determine $\hat{\mathbf{F}}_{ij}$ from the singular vector corresponding to the smallest singular value of $\widehat{\mathbf{A}}$, where $\widehat{\mathbf{A}}$ is composed from the matches $\hat{\mathbf{x}}^{(i)} \leftrightarrow \hat{\mathbf{x}}^{(j)}$ as explained in equation 2.51.

b) **Constraint enforcement**: Replace $\hat{\mathbf{F}}_{ij}$ by $\hat{\mathbf{F}}'_{ij}$ such that $\det(\hat{\mathbf{F}}'_{ij}) = 0$. In other words enforce the constraint $\operatorname{rank}(\hat{\mathbf{F}}_{ij}) = 2$ by computing the Singular Value Decomposition of F:

$$\hat{\mathbf{F}}_{ij} = \mathbf{U}_{\mathrm{F}} \mathbf{D}_{\mathrm{F}} \mathbf{V}_{\mathrm{F}}^{\mathrm{T}}$$

and then setting the smallest singular value of $\hat{\mathbf{F}}_{ij}$ equal to 0 and name it as $\mathbf{D}'_{\mathrm{F}}$. So the new estimate for $\hat{\mathbf{F}}_{ij}$, $\hat{\mathbf{F}}'_{ij}$ is given by

$$\hat{\mathbf{F}}'_{ij} = \mathbf{U}_{\mathrm{F}} \mathbf{D}'_{\mathrm{F}} \mathbf{V}_{\mathrm{F}}^{\mathrm{T}}$$

(iii) **Denormalization:** Set $\mathbf{F}_{ij} = (\mathbf{T}_j)^{\mathrm{T}} \hat{\mathbf{F}}'_{ij} \mathbf{T}_i$. Matrix $\mathbf{F}_{ij}$ is the fundamental matrix corresponding to the original data $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$.

Algorithm 2.7. *normalized 8-point algorithm [1]*

2.2.5 Singular Value Decomposition

The aim for this section is to lay some mathematical basis for SVD analysis which is useful in Principal Component Analysis (PCA) when the dimensions for covariance matrix are reduced using PCA. SVD analysis is also used in parts of the proposed methodology for solving linear set of equations.

Definition:

Any real *mxn* matrix **A** can be decomposed uniquely as:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{2.52}$$

U is *mxn* and column orthogonal (its columns are eigenvectors of $\mathbf{A}\mathbf{A}^T$):

$$\mathbf{A}\mathbf{A}^T = \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}\mathbf{D}\mathbf{U}^T = \mathbf{U}\mathbf{D}^2\mathbf{U}^T \tag{2.53}$$

V is *nxn* and orthogonal (its columns are eigenvectors of $\mathbf{A}^T\mathbf{A}$)

$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{V}\mathbf{D}^2\mathbf{V}^T \tag{2.54}$$

D is *nxn* diagonal (non-negative real values called singular values)

$$\mathbf{D} = diag(\sigma_1, \sigma_2, \cdots, \sigma_n)$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ (if σ is a singular value of **A**, it's square is an eigenvalue of $\mathbf{A}^T\mathbf{A}$).

If $\mathbf{U} = (u_1\ u_2\ \cdots u_n)$ and $= (v_1\ v_2\ \cdots v_n)$, then

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i u_i v_i^T \tag{2.55}$$

The column of **U** are called the *left singular vector* and the row of $\mathbf{V}^T$ are *right* singular vector.

A list of possible applications for SVD is provided below:

- ***Computing the rank using SVD***

The rank of a matrix is equal to the number of non-zero singular values.

- ***Computing the inverse of a matrix using SVD***

A square matrix *A* is nonsingular if $\sigma_i \neq 0\ \forall i$

If **A** is a *nxn* non-singular matrix, then its inverse is given by

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \ \Rightarrow\ \mathbf{A}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T \tag{2.56}$$

where

$$\mathbf{D}^{-1} = diag(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \cdots, \frac{1}{\sigma_n})$$

If **A** is singular or ill-conditioned, then we can use SVD to approximate its inverse by the following matrix:

$$\mathbf{A}^{-1} = (\mathbf{U}\mathbf{D}\mathbf{V}^T)^{-1} \approx \mathbf{V}\mathbf{D}_0^{-1}\mathbf{U}^T \tag{2.57}$$

where

$$\mathbf{D}_0^{-1} = \begin{cases} {}^{1}/_{\sigma_i} & if\ \sigma_i > t \\ 0 & otherwise \end{cases}$$

- ***The condition of a matrix***

Consider the system of linear equations

$$\mathbf{Ax} = \mathbf{b} \tag{2.58}$$

If small changes in **b** can lead to relatively large changes in the solution **x**, then we call this system as ill-conditioned.

The ratio given below is related to the condition of **A** and measures the degree of singularity of **A** (the larger this value is, the closer **A** is to being singular)

$${}^{\sigma_1}/_{\sigma_n} \tag{2.59}$$

- ***Least Squares Solution of mxn systems***

Consider equation 2.58 again by considering **A** as *m*×*n* matrix with $m > n$. In such conditions, the set of equations are known as the over-determined system of linear equations. Let *r* be the residual vector for some *x*:

$$\mathbf{r} = \mathbf{Ax} - \mathbf{b} \tag{2.60}$$

The vector $\mathbf{x}^*$ which yields the smallest possible residual is called a least square solution (it is an approximate solution)

$$\|\mathbf{r}\| = \|\mathbf{Ax}^* - \mathbf{b}\| \leq \|\mathbf{Ax} - \mathbf{b}\|, \mathbf{x}\epsilon\mathbb{R}^n \tag{2.61}$$

Although a least square solution always exists, it might not be unique.

The least square solution **x** with smallest form $\|\mathbf{x}\|$ is unique and it is given by:

$$\mathbf{A}^T\mathbf{A} = \mathbf{A}^T\mathbf{b} \quad or \quad \mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{A}^+\mathbf{b} \tag{2.62}$$

- ***Computing $\mathbf{A}^+$ using SVD***

If $\mathbf{A}^T\mathbf{A}$ is ill-conditioned or singular, we can use SVD to obtain a least squares solution as follows:

$$\mathbf{x} = \mathbf{A}^+\mathbf{b} \approx \mathbf{V}\mathbf{D}_0^{-1}\mathbf{U}^T\mathbf{b} \tag{2.63}$$

where

$$\mathbf{D}_0^{-1} = \begin{cases} {}^{1}/_{\sigma_i} & if\ \sigma_i > t \\ 0 & otherwise \end{cases}$$

- ***Homogeneous System***

Suppose $\mathbf{b} = 0$, then the linear system is called homogeneous, with **A** is *mxn* and $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$*:*

$$\mathbf{Ax} = 0 \tag{2.64}$$

The minimum-norm solution in this case is $\mathbf{x} = 0$.

For homogeneous linear system, the meaning of least squares solution is modified by imposing the constraint:

$$\|\mathbf{x}\| = 1 \tag{2.65}$$

This is a "*constrained*" optimization problem:

$$min_{||\mathbf{x}||=1}\|\mathbf{A}\mathbf{x}\|$$

The minimum-norm solution for homogeneous system is not always unique.

Special case: $rank(\mathbf{A}) = n - 1$ $(m \geq n-1, \sigma_n = 0)$

solution is $\mathbf{x} = a\mathbf{v}_n$ (a is a constant, $\mathbf{v}_n$ is the last column of $\mathbf{V}$ corresponding to the smallest σ).

General case: $rank(\mathbf{A}) = n - k$ $(m \geq n-k, \sigma_{-k+1n} = \sigma_n = 0)$

solution is $\mathbf{x} = a_1\mathbf{v}_{n-k} + a_2\mathbf{v}_{n-k-1} + \cdots + a_k\mathbf{v}_n$ (a_i is a constant) with

$$a_1^2 + a_2^2 + \cdots + a_n^2 = 1$$

These explanations have been obtained from [5].

2.2.6 Camera Matrices

According to section 2.2.1 for camera parameters, by combining extrinsic with intrinsic camera parameters and rewriting the final equation in matrix form, the so called camera matrix **P** is derived. As mentioned before, this matrix may be decomposed into matrix multiplication of a few matrices containing the intrinsic and extrinsic parameters and in the case of radial lens distortion, a few additional parameters for radial distortion.

Here, we aim to represent the camera model and its parameters in closed-form equations expressed by matrices. Finally, we get to a single 3×4 matrix that maps the points from the 3D world coordinate system to the 2D image coordinate system (both expressed in homogeneous coordinates) and is named as camera matrix. According to section 2.2.1, one can derive two matrices for intrinsic and extrinsic parameters separately, as below:

$$\mathbf{P}_{in} = \begin{bmatrix} -f/s_x & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}_{ex} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -\mathbf{R}_1^T\mathbf{t} \\ r_{21} & r_{22} & r_{23} & -\mathbf{R}_2^T\mathbf{t} \\ r_{31} & r_{32} & r_{33} & -\mathbf{R}_3^T\mathbf{t} \end{bmatrix} \tag{2.66}$$

Right away, using homogeneous coordinates, we get:

$$\begin{bmatrix} x_h \\ y_h \\ w \end{bmatrix} = \mathbf{P}_{in}\mathbf{P}_{ex}\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \mathbf{P}\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.67}$$

where **P** is called the projection or camera matrix (*3x4*).

Finally, by inhomogenization, the 2D pixel coordinates will be:

$$\begin{cases} x_{im} = \frac{x_h}{w} = \frac{p_{11}x_w + p_{12}y_w + p_{13}z_w + p_{14}}{p_{31}x_w + p_{32}y_w + p_{33}z_w + p_{34}} \\ - - - - - - - - - - - - - - - - \\ y_{im} = \frac{y_h}{w} = \frac{p_{21}x_w + p_{22}y_w + p_{23}z_w + p_{24}}{p_{31}x_w + p_{32}y_w + p_{33}z_w + p_{34}} \end{cases} \quad (2.68)$$

Note: the relation of 3D points and their 2D projections can be seen as a linear transformation from the projective space $(x_w, y_w, z_w, 1)^{\mathrm{T}}$, to the projective plane $(x_h, y_h, w)^{\mathrm{T}}$.

There are different ways for simplifying such projection or camera matrix. The basic idea is to make some simplifying assumptions in the camera model which results in removing the effect of some intrinsic or extrinsic camera parameters. Within each of these methods, the camera model is named differently. Some of such simplifications are discussed in the following paragraphs.

❖ **The perspective camera model**

By assuming $o_x, o_y = 0$ and $s_x, s_y = 1$ (The camera image plane is placed right in front of the focal point and there is no scale change in x and y directions on the camera image plane):

$$\mathbf{P} = \begin{bmatrix} -fr_{11} & -fr_{12} & -fr_{13} & f\mathbf{R}_1^T\mathbf{t} \\ -fr_{21} & -fr_{22} & -fr_{23} & f\mathbf{R}_2^T\mathbf{t} \\ r_{31} & r_{32} & r_{33} & -\mathbf{R}_3^T\mathbf{t} \end{bmatrix} \quad (2.69)$$

One may verify the correctness of the above matrix by:

$$\mathbf{x} = \mathbf{PX} = \begin{bmatrix} -f\mathbf{R}_1^T & f\mathbf{R}_1^T\mathbf{t} \\ -f\mathbf{R}_2^T & f\mathbf{R}_2^T\mathbf{t} \\ \mathbf{R}_3^T & -\mathbf{R}_3^T\mathbf{t} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} -f\mathbf{R}_1^T(\mathbf{X} - \mathbf{t}) \\ -f\mathbf{R}_2^T(\mathbf{X} - \mathbf{t}) \\ \mathbf{R}_3^T(\mathbf{X} - \mathbf{t}) \end{bmatrix} \quad (2.70)$$

By inhomogenization, the 2D pixel coordinates will be:

$$\Rightarrow x = -f\frac{\mathbf{R}_1^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} \qquad y = -f\frac{\mathbf{R}_2^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} \quad (2.71)$$

which is the same as the result in section 2.2.1 (from camera to image coordinates).

❖ **The weak perspective camera model**

For reducing the perspective distortion of object images, sometimes the camera model is assumed to be of this type. This model is especially true for the case of objects being far from the camera, so that the effect of perspectivity is decreased. This model is described in the following form:

$$\mathbf{P} = \begin{bmatrix} -fr_{11} & -fr_{12} & -fr_{13} & f\mathbf{R}_1^T\mathbf{t} \\ -fr_{21} & -fr_{22} & -fr_{23} & f\mathbf{R}_2^T\mathbf{t} \\ r_{31} & r_{32} & r_{33} & \mathbf{R}_3^T(\overline{\mathbf{X}} - \mathbf{t}) \end{bmatrix} \tag{2.72}$$

where $\overline{\mathbf{X}}$ is the centroid of the object (i.e., object's average distance from the camera). One can again verify the correctness of the above matrix by:

$$\mathbf{x} = \mathbf{PX} = \begin{bmatrix} -f\mathbf{R}_1^T & f\mathbf{R}_1^T\mathbf{t} \\ -f\mathbf{R}_2^T & f\mathbf{R}_2^T\mathbf{t} \\ \mathbf{0}^T & \mathbf{R}_3^T(\overline{\mathbf{X}} - \mathbf{t}) \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} -f\mathbf{R}_1^T(\mathbf{X} - \mathbf{t}) \\ -f\mathbf{R}_2^T(\mathbf{X} - \mathbf{t}) \\ \mathbf{R}_3^T(\mathbf{X} - \mathbf{t}) \end{bmatrix} \tag{2.73}$$

By inhomogenization, the 2D pixel coordinates will be:

$$\Rightarrow x = -f\frac{\mathbf{R}_1^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} \qquad y = -f\frac{\mathbf{R}_2^T(\mathbf{X}-\mathbf{t})}{\mathbf{R}_3^T(\mathbf{X}-\mathbf{t})} \tag{2.74}$$

❖ **The affine camera model**

In this type, all the entries of the projection matrix are totally unconstrained except the first 3 elements of the last row which accounts for perspectivity:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & p_{34} \end{bmatrix} \tag{2.75}$$

As the effect of perspectivity is completely neglected in this model, this model does not appear to correspond to any physical camera. Such model is only useful for simplifying equations and appealing geometric properties. The affinity of this model leads to preserving parallelism while not preserving angles. The concept of affinity has been discussed in more details in section 2.1.5.

❖ **The simplified camera model**

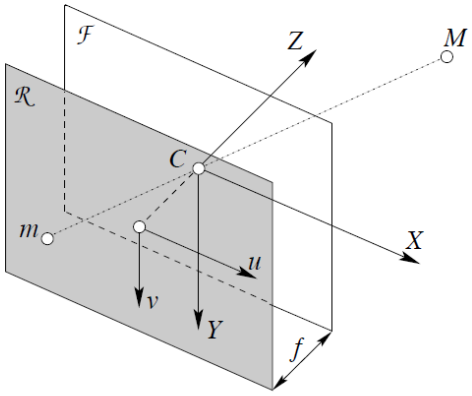


Figure 2.29: *Camera geometric model*

In the simplified camera geometric model, **M** is defined as $[X\ Y\ Z]$ in 3-space and (u,v) for its projection in the image plane, as shown in figure 2.29.

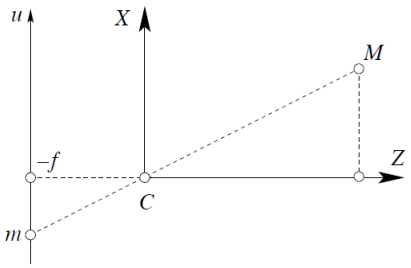


Figure 2.30: *Triangle similitude in the camera geometric model*

For the triangle similitude shown in figure 2.30, we can write:

$$\frac{f}{z} = \frac{-u}{x} = \frac{-x}{y} \tag{2.76}$$

or

$$\begin{cases} u = \frac{-f}{z}x \\ v = \frac{-f}{z}y \end{cases} \tag{2.77}$$

Such mapping from **M** to **m** is called perspective projection. This 3D to 2D transformation is not essentially linear, but if we utilize the homogeneous coordinates it will become linear. So in homogeneous coordinates, we can write:

$$\mathbf{m} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \mathbf{M} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.78}$$

With 1 as the fourth element in **M**, we exclude the infinite point and the perspective projection is rewritten as:

$$z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} -fx \\ -fy \\ z \end{pmatrix} = \begin{pmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.79}$$

or in matrix notation:

$$\mathbf{m} = \lambda \mathbf{K}\mathbf{M} \tag{2.80}$$

where **K** is the matrix which holds intrinsic camera parameters (in this case only *f*)

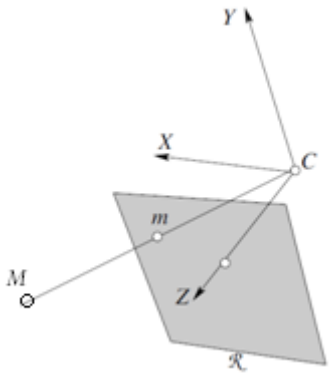


Figure 2.31: *Perspective projection with the image plane in front of the centre of the camera*

The last consideration is that **P** with $f = -1$ corresponds to $\mathbf{P} = [I|0]$ a special case in which the image plane is in front of the center of the projection (see figure 2.31).

From now up to end of this thesis, we will consider the perspective camera model, since it has more degrees of freedom and capable of modelling a variety of transformations.

2.2.7 2D Planar Homography

In this section, the concept of 2D planar homography is introduced (denoted by $\mathbf{H}_\pi$). The problem is visualized in figure 2.32. The 2D planar homography $\mathbf{H}_\pi$ is simply a function which maps the points in one view to another. This is a projective relation since it depends only on the intersections of planes with lines. The main difference between such 2D mapping function and the other 2D transformation models discussed in section 2.1.5 is that 2D planar homography is derived based on having some information from the 3D world that is the epipolar plane. This plane is shown in figure 2.32.

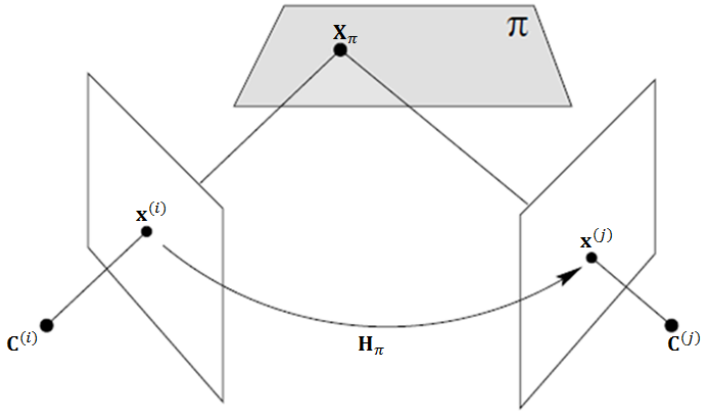


Figure 2.32: *Homography Induced by the epipolar plane π. This figure is taken from [1].*

As shown in figure 2.32, the planar homography maps a point $\mathbf{x}^{(i)}$ in one image via the plane π to its correspondent point $\mathbf{x}^{(j)}$ in the second image. In other words, under such mapping, the transformation of points in 3D from one camera to another can be written as:

$$\mathbf{X}^{(j)} = \mathbf{H}_\pi \mathbf{X}^{(i)} \qquad \mathbf{X}^{(i)}, \mathbf{X}^{(j)} \in \mathbb{R}^3 \tag{2.81}$$

On the epipolar plane π, using homogenous coordinates, we have

$$\lambda_1 \mathbf{x}^{(i)} = \mathbf{X}^{(i)}, \lambda_2 \mathbf{x}^{(j)} = \mathbf{X}^{(j)} \Rightarrow \lambda_2 \mathbf{x}^{(j)} = \mathbf{H}_\pi \lambda_1 \mathbf{x}^{(i)} \Rightarrow \mathbf{x}^{(j)} \sim \mathbf{H}_\pi \mathbf{x}^{(i)} \tag{2.82}$$

The equation above tells us that $\mathbf{x}^{(j)}$ is equal to $\mathbf{H}_\pi \mathbf{x}^{(i)}$ up to a scale (due to universal scale ambiguity). Yet, there is another way to define the planar homography which is:

$$\mathbf{x}^{(i)} = \mathbf{H}_{1\pi} \mathbf{X}_\pi \tag{2.83}$$

$$\mathbf{x}^{(j)} = \mathbf{H}_{2\pi} \mathbf{X}_\pi \tag{2.84}$$

Equation (2.83) expresses the mapping from the first image plane to the world plane π. Similarly, equation (2.84) expresses the mapping from the second image plane to the world plane π. The composition of these two mappings will result in a homography between the two image planes as follows:

$$\mathbf{x}^{(j)} = \mathbf{H}_{2\pi}\mathbf{H}_{1\pi}^{-1}\mathbf{x}^{(i)} = \mathbf{H}_{\pi}\mathbf{x}^{(i)} \tag{2.85}$$

where

$$\mathbf{H}_{\pi} = \mathbf{H}_{2\pi}\mathbf{H}_{1\pi}^{-1} \tag{2.86}$$

Basically, there are two types of relationship between two views of a scene:

1. Through the epipolar geometry: (refer to section 2.2.3) where a point in one view determines a line in the other which is the image of the ray through that point;
2. Through the homography: where a point in one view determines a point in the other which is the image of the intersection of the ray with the world plane. In such case, it is said that the world plane induces a homography between the views.

Such definition for induced homography can be mathematically derived given the projection matrices for the two cameras in the canonical form as:

$$\mathbf{P}^{(1)} = [I \mid 0] \quad ; \quad \mathbf{P}^{(2)} = [A \mid a] \tag{2.87}$$

and a plane π defined by $\pi^T\mathbf{X} = 0$ with coordinates $\pi = (\mathbf{v}^T, 1)^T$ where $\mathbf{v}^T$ is the orthonormal vector for the plane π. Then the homography induced by the plane will be:

$$\mathbf{x}^{(2)} = \mathbf{H}_{\pi}\mathbf{x}^{(1)} \tag{2.88}$$

with

$$\mathbf{H}_{\pi} = \mathbf{A} - \mathbf{a}\mathbf{v}^T \tag{2.89}$$

The proof for this formulation will be shortly explained here. In fact, to compute $\mathbf{H}_{\pi}$ we back-project a point $\mathbf{x}^{(1)}$ in the first view to 3D world coordinate system and determine the intersection ($\mathbf{X}$) of the corresponding ray with the plane π. The 3D point $\mathbf{X}$ satisfies $\pi^T\mathbf{X} = 0$ and can be rewritten as:

$$\mathbf{X} = \left((\mathbf{x}^{(1)})^T, -\mathbf{v}^T\mathbf{x}^{(1)}\right)^T \tag{2.90}$$

This 3D point is then projected onto the second view:

$$\mathbf{x}^{(2)} = \mathbf{P}^{(2)}\mathbf{X} = [\mathbf{A}\,|\mathbf{a}]\mathbf{X}$$
$$= \mathbf{A}\mathbf{x}^{(1)} - \mathbf{a}\mathbf{v}^T\mathbf{x}^{(1)} = (\mathbf{A} - \mathbf{a}\mathbf{v}^T)\mathbf{x}^{(1)} \qquad (2.91)$$

as required for the equation (2.89).

The homography is compatible with epipolar geometry, i.e. a set of point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ that defines a homography $\mathbf{H}_\pi$, also obey the epipolar constraint $(\mathbf{x}^{(j)})^T\mathbf{F}_{ij}\mathbf{x}^{(i)} = 0$. In other words, the homography $\mathbf{H}_\pi$ is said to be consistent or compatible with $\mathbf{F}_{ij}$. Since $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ obey the epipolar constraint if $\mathbf{H}_\pi$ is induced by a plane, then from the epipolar constraint

$$(\mathbf{x}^{(j)})^T\mathbf{F}_{ij}\mathbf{x}^{(i)} = \left(\mathbf{H}_\pi\mathbf{x}^{(i)}\right)^T\mathbf{F}_{ij}\mathbf{x}^{(i)} = (\mathbf{x}^{(i)})^T\mathbf{H}_\pi^T\mathbf{F}_{ij}\mathbf{x}^{(i)} = 0 \qquad (2.92)$$

Since equation 2.41 holds for all $\mathbf{x}^{(i)}$ and according to interesting properties of skew-symmetric matrices, $\mathbf{H}_\pi^T\mathbf{F}_{ij}$ should be skew-symmetric. In other words, a homography $\mathbf{H}_\pi$ is compatible with a fundamental matrix $\mathbf{F}_{ij}$ if and only if the matrix $\mathbf{H}_\pi^T\mathbf{F}_{ij}$ is skew-symmetric. Hence, due to symmetric properties in the elements of an skew-symmetric matrix, the following equation should hold which is also known as compatibility constraint:

$$\mathbf{H}_\pi^T\mathbf{F}_{ij} + \mathbf{F}_{ij}^T\mathbf{H}_\pi = 0 \qquad (2.93)$$

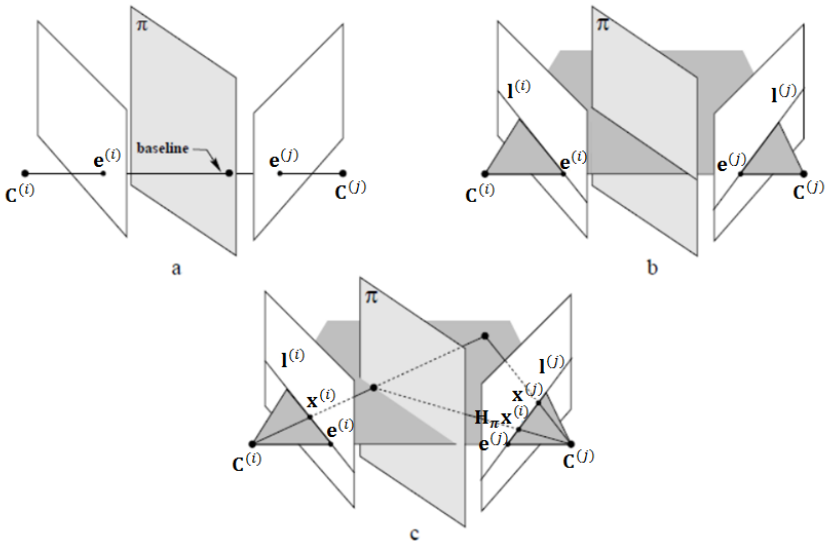


Figure 2.33: *The homography induced by a plane is coupled to the epipolar geometry and satisfies constraints. This figure is taken from [1].*

In Figure 2.33, the usage of several relationships between points and lines for satisfying equation 2.92 are illustrated. In part **a** of this figure, one epipole $\mathbf{e}^{(i)}$ is mapped by the homography $\mathbf{H}_\pi$ to $\mathbf{e}^{(j)}$:

$$\mathbf{e}^{(j)} = \mathbf{H}_\pi \mathbf{e}^{(i)} \tag{2.94}$$

since the epipoles are images of the point on the plane where the baseline intersects plane π. In part **b** of this figure, epipolar line $\mathbf{l}^{(i)}$ is mapped by the homography $\mathbf{H}_\pi$ to $\mathbf{l}^{(j)}$:

$$\begin{cases} \mathbf{l}^{(i)} = \mathbf{H}_\pi^T \mathbf{l}^{(j)} \\ \mathbf{l}^{(j)} = \mathbf{H}_\pi \mathbf{l}^{(i)} \end{cases} \tag{2.95}$$

Finally in part **c** of this figure, any point $\mathbf{x}^{(i)}$ mapped by the homography $\mathbf{H}_\pi$ lies on its corresponding epipolar line $\mathbf{l}^{(j)}$:

$$\mathbf{l}^{(j)} = \mathbf{F}_{ij}\mathbf{x}^{(i)} = \mathbf{x}^{(j)} \times (\mathbf{H}_\pi \mathbf{x}^{(i)}) \tag{2.96}$$

To conclude the discussion about the 2D planar homography, an explicit expression for a homography $\mathbf{H}_\pi$ induced by a plane using the constraint in equation 2.97 is given. Given the fundamental matrix $\mathbf{F}_{ij}$ between two views, the three-parameter family of homography induced by a world plane π is:

$$\mathbf{H}_\pi = \boldsymbol{A} - \mathbf{e}^{(j)}\mathbf{v}^T \tag{2.97}$$

where $\left[\mathbf{e}^{(j)}\right]_x \mathbf{A} = \mathbf{F}$ is any decomposition of the fundamental matrix and $\left[\mathbf{e}^{(j)}\right]_x$ is a skew-symmetric *3×3* matrix corresponding the vector $\mathbf{e}^{(j)}$. This notation is described in section 2.2.4. By the explanations presented on the concept of 2D planar homography, here we introduce three computational methods for estimating such 2D transformation.

The 1st method: the input to the algorithm is simply the image pair and the output is the estimated homography together with a better set of point correspondences consistent with the homography itself. This is a general algorithm for the estimation of homography. Since, a set of point correspondences from SIFT/SURF followed by RANSAC are made and discussed in previous sections, we skip the first 3 steps of the algorithm and continue from the steps 4 and 5, except one important note that is about step 1. The minimum number of samples should be at least 4, since 4 point correspondences uniquely determine a 2D planar homography. The last 2 steps

in this method are the robust estimation and guided matching process. The aim of this final stage is twofold:

1. Obtain an improved estimate of the homography by using all the inliers in the estimation;
2. Obtain more inlying matches from the putative correspondence set because a more accurate 2D planar homography is available.

Here, we only describe the last two steps of the algorithm 2.8 in more details, since it is assumed that the point correspondences are correct, so that the focus will be on the methodology for computing $\mathbf{H}_\pi$.

Optimal estimation: The idea is quite the same as in RANSAC, and in particular the distance measured for inliers described in 3.1.3. In this case in fact the homography estimated, $\widehat{\mathbf{H}}_\pi$ itself is a component to derive the cost function. The process re-estimates $\mathbf{H}_\pi$ from all correspondences classified as inliers, by minimizing the ML cost function which is:

$$\sum_i d\left(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}\right)^2 + d\left(\mathbf{x}^{(j)}, \hat{\mathbf{x}}^{(j)}\right)^2 = d\left(\mathbf{x}^{(i)}, \mathbf{H}_\pi^{-1}\mathbf{x}^{(j)}\right)^2 + d\left(\mathbf{x}^{(j)}, \mathbf{H}_\pi\mathbf{x}^{(i)}\right)^2 \qquad (2.98)$$

where $d(x, \hat{x})$ is the notation for the Euclidean distance between 2 points. Minimizing this cost function involves determining both $\widehat{\mathbf{H}}_\pi$ and a set of subsidiary correspondences $\{\hat{\mathbf{x}}^{(i)}\}$ and $\{\hat{\mathbf{x}}^{(j)}\}$. This estimation models, for example the situation that the measured correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ arise from images of points on a world plane and the goal is to estimate a point on the world plane $\widehat{\mathbf{X}}$ from $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ which is then re-projected to the point correspondence set $\hat{\mathbf{x}}^{(i)} \leftrightarrow \hat{\mathbf{x}}^{(j)}$ accurately (in an ideal case), as shown in figure 3.34:

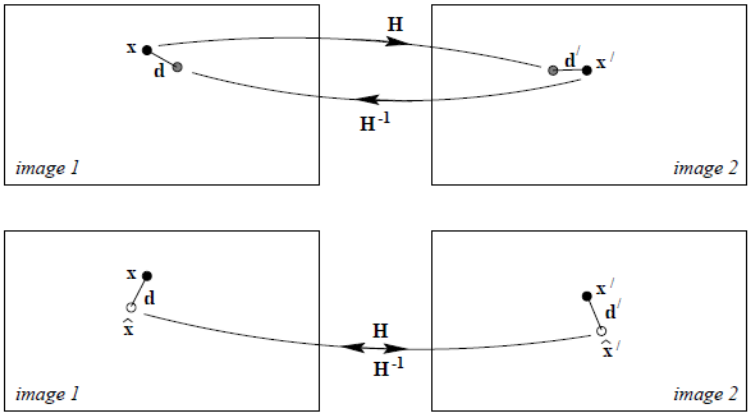


Figure 2.34: *A comparison between symmetric transfer error (upper) and re-projection error (lower) when estimating a homography. This figure is taken from [1].*

Under an estimated 2D planar homography $\widehat{\mathbf{H}}_{\pi}$, the points $\mathbf{x}^{(j)}$ and $\widehat{\mathbf{H}}_{\pi}\mathbf{x}^{(i)}$ do not correspond perfectly, and neither do the points $\mathbf{x}^{(i)}$ and $\widehat{\mathbf{H}}_{\pi}^{-1}\mathbf{x}^{(j)}$. However, the estimated points, $\hat{\mathbf{x}}^{(i)}$ and $\hat{\mathbf{x}}^{(j)}$ do correspond perfectly by the 2D planar homography $\hat{\mathbf{x}}^{(j)} = \widehat{\mathbf{H}}_{\pi}\hat{\mathbf{x}}^{(i)}$ (equation 2.87).

The re-projection error function in equation 2.98 can be iteratively minimized until the number of correspondence becomes stable. The algorithm for minimizing such cost function is the Levenberg-Marquardt iteration or LMA method that is a slight variant of the Gauss-Network algorithm (GNA). Like GNA, this method provides a numerical solution to the problem of minimizing a function, generally nonlinear, over a space of parameters of the function. The LMA is more robust that the GNA, which means that in many cases it finds a solution even if it starts very far off the final minimum. Finally, LMA is a very popular curve fitting algorithm. However, there is always a risk of being trapped by only a local minimum, and not a global minimum with respect to the global function values.

Input: 2 images $\mathbf{I}_i$ and $\mathbf{I}_j$.

Output: 3x3 matrix $\mathbf{H}_{ij}$.

1. **Interest points**: compute interest points in individual images from a pair of cameras.
2. **Putative correspondences**: compute a set of interest point matches based on proximity and similarity of their intensity neighborhood.
3. **RANSAC robust estimation**: From an initial set of (n+k) point correspondences, n true point correspondences will be extracted as subset. At the end of this step, H with the largest set of inliers n is estimated (by the second or third method, mentioned below) and chosen.
4. **Optimal estimation**: re-estimate $\mathbf{H}_\pi$ from all correspondences classified as inliers, by minimizing the ML cost function (equation 2.98) using the Levenberg-Marquardt algortithm (LMA).
5. **Guided matching**: Further interest point correspondences are now determined using the estimated $\mathbf{H}_\pi$ to define a search region about the transferred point position.

Algorithm 2.8. *1st method for estimation of a 2D planar homography (with 4 points)*

The 2nd method: the homography induced by a world plane π between two images will be calculated given some information about the epipolar geometry. In order to estimate the homography $\mathbf{H}_\pi$ induced by image planes, given the point correspondences, one should compute the fundamental matrix $\mathbf{F}_{ij}$ with the most reliable point correspondences satisfying the equation 2.42 and the following condition (Singularity of Fundamental Matrix):

$$det(\mathbf{F}_{ij}) = 0 \tag{2.99}$$

After computing $\mathbf{F}_{ij}$ estimation of the 2D planar homography is done according to Algorithm 2.9 as follows:

Input: $\mathbf{F}_{ij}$ and 3 non-collinear point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ which are the images of 3D point **X**.

Output: 3x3 matrix $\mathbf{H}_\pi$ such that $\mathbf{x}^{(j)} = \mathbf{H}_\pi \mathbf{x}^{(i)}$ induced by the plane of **X**.

(i) Choose $\mathbf{A} = \left[\mathbf{e}^{(i)}\right]_x \mathbf{F}_{ij}$ and solve linearly **v** from $\mathbf{Mv} = \mathbf{b}$ where and **M** is a 3x3 matrix with rows $\mathbf{x}_i^T$ and **b** is a 3-vector with the following elements:

$$b_i = \left(\mathbf{x}^{(j)} \times (\mathbf{A}x_i)\right)^T \left(\mathbf{x}^{(j)} \times \mathbf{e}^{(j)}\right) / \left\|\mathbf{x}^{(j)} \times \mathbf{e}^{(j)}\right\|^2$$

(ii) Finally, $\mathbf{H}_\pi = \mathbf{A} - \mathbf{e}^{(j)}\mathbf{v}^T$

Algorithm 2.9. *2nd method for estimation of 2D planar homography (with 3 points)*

Thus, given $\mathbf{F}_{ij}$ and 3 non-collinear point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$, the homography induced by the plane of the 3D point $\mathbf{X}_\pi$ is given as:

$$\mathbf{H}_\pi = \mathbf{A} - \mathbf{e}^{(j)}(\mathbf{M}^{-1}\mathbf{b})^T \tag{2.100}$$

The 3rd method: is the most simple and trivial way without considering the epipolar geometry and 3D information about the 3D object. We start from equation $\mathbf{x}^{(j)} \sim \mathbf{H}_\pi \mathbf{x}^{(i)}$. In this method, we don't need to assume epipolar geometry and we only rely on point correspondences. In homogeneous coordinates, we would get the following set of equations according to the definition of homography:

$$\begin{bmatrix} x^{(j)} \\ y^{(j)} \\ z^{(j)} \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x^{(i)} \\ y^{(i)} \\ z^{(i)} \end{bmatrix} \Leftrightarrow \mathbf{x}^{(j)} = \mathbf{H}_\pi \mathbf{x}^{(i)} \tag{2.101}$$

Then, by going back to inhomogeneous coordinates ($x'^{(j)} = x^{(j)}/z^{(j)}$, $y'^{(j)} = y^{(j)}/z^{(j)}$), it is clear that:

$$\begin{cases} x'(j) = \dfrac{H_{11}x^{(i)} + H_{12}y^{(i)} + H_{13}z^{(i)}}{H_{31}x^{(i)} + H_{32}y^{(i)} + H_{33}z^{(i)}} \\ y'(j) = \dfrac{H_{21}x^{(i)} + H_{22}y^{(i)} + H_{23}z^{(i)}}{H_{31}x^{(i)} + H_{32}y^{(i)} + H_{33}z^{(i)}} \end{cases} \tag{2.102}$$

Without loss of generality, we can set $z_1 = 1$ and rearrange:

$$\begin{cases} x'^{(j)}\left(H_{31}x^{(i)} + H_{32}y^{(i)} + H_{33}z^{(i)}\right) = H_{11}x^{(i)} + H_{12}y^{(i)} + H_{13} \\ y'^{(j)}\left(H_{31}x^{(i)} + H_{32}y^{(i)} + H_{33}z^{(i)}\right) = H_{21}x^{(i)} + H_{22}y^{(i)} + H_{23} \end{cases} \tag{2.103}$$

The set of equations in (2.103) are linear in the unknown elements of $\mathbf{H}_\pi$ and can be rearranged again to get the following closed form:

$$\begin{cases} \mathbf{a}_x^T \mathbf{h} = 0 \\ \mathbf{a}_y^T \mathbf{h} = 0 \end{cases} \tag{2.104}$$

where

$$\boldsymbol{h} = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T$$

$$\mathbf{a}_x = (-x_1, -y_1, -1, 0, 0, 0, x'_2 x_1, x'_2 y_1, x'_2)$$

$$\mathbf{a}_y = (0, 0, 0, -x_1, -y_1, -1, y'_2 x_1, y'_2 y_1, y'_2)$$

Since $\mathbf{H}_\pi$ has 9 unknown elements and 1 could be skipped as the scale factor and due to 2 equations from each point correspondence, 4 point correspondences will be enough for the following set of linear equations to be solved for **h**.

$$\mathbf{Ah} = 0 \tag{2.105}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{x1} \\ a_{y1} \\ \vdots \\ a_{xN} \\ a_{yN} \end{bmatrix}$$

For solving this system of equations, we first compute the SVD of **A** (For more details on SVD, refer to section 2.2.5):

$$\mathbf{A} = \mathbf{U\Sigma V}^T = \sum_{i=1}^{9} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{2.106}$$

When performed in MATLAB, the singular values σ_i are sorted in the descending order so that σ_9 holds the smallest singular value of **A**. We are interested in 2 cases according to the value of σ_9:

1. If the 2D planar homography $\mathbf{H}_\pi$ is *exactly determined*, then $\sigma_9 = 0$, and there exists a homography $\mathbf{H}_\pi$ that fits the points exactly.
2. If the 2D planar homography $\mathbf{H}_\pi$ is *overdetermined*, then $\sigma_9 > 0$. Here σ_9 represents a residual or goodness of fit.

Finally, the right singular vector (a column from **V**) which corresponds to the smallest singular value, σ_9 is chosen as the solution, **h**, which contains the unknown elements of the 2D planar homography matrix $\mathbf{H}_\pi$ that best fits the points.

As we are interested in using the epipolar geometry within multiple views, algorithm 2.9 is mainly used in this thesis.

2.2.8 Triangulation

The purpose for this section is to introduce a method for computing the position of a point **X** in 3D world coordinate system given its two projections $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ on the camera image planes and the corresponding camera matrices $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$.

It is assumed that there exist errors only in the measured image coordinates, not in the camera projection matrices $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$. The computation method for camera matrices will be described later in the section 3.4.

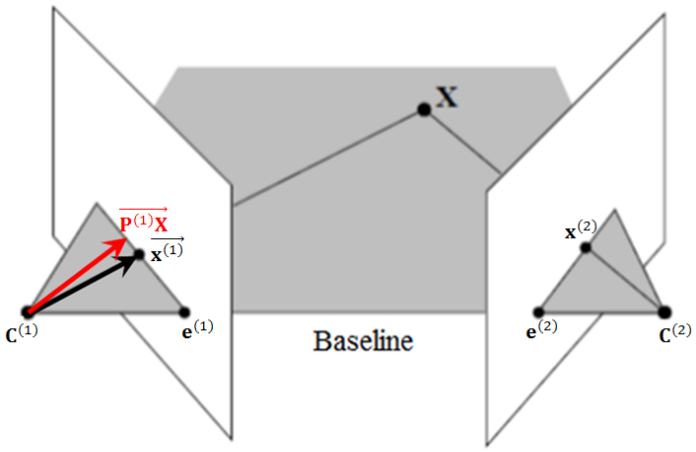


Figure 2.35: *Triangulation. The image points* $\boldsymbol{x^{(1)}}$ *and* $\boldsymbol{x^{(2)}}$ *back project to rays. If the epipolar constraint* $(\boldsymbol{x^{(2)}})^{\boldsymbol{T}}\boldsymbol{F_{12}}\boldsymbol{x^{(1)}} = 0$ *is satisfied, then these two rays lie in a plane, and so intersect in a point* **X** *in 3-space. This figure is partly taken from [1].*

As showed in the figure 2.35, triangulation refers to the problem of estimating the optimal position of a point **X** in 3-space given its 2 or more projections $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ on some camera image planes. This problem could be considered as a simple minimization problem, in which a cost function is minimized. This is especially critical in affine and projective reconstruction in which there is no meaningful metric information about the object space (i.e. uncalibrated case faced in this thesis). In the following paragraphs we will describe the main steps of this process.

- ✓ **Note:** In order to do triangulation for finding **X** given its multiple projections, **F** and the corresponding camera matrices should be provided as a priori.

The ray $\overline{C^{(1)}\mathbf{X}}$ will form a plane with the baseline $\overline{C^{(1)}C^{(2)}}$ as shown in figure 2.35. In addition, the ray $\overline{C^{(2)}\mathbf{X}}$ will form another plane with the baseline. For these two planes to be the same, the epipolar constraint should be satisfied. For simple proof and

explanations, refer to chapter 9 of [1]. In such case, the two rays $\overline{C^{(1)}\mathbf{X}}$ and $\overline{C^{(2)}\mathbf{X}}$ will intersect at some point $\mathbf{X}$ in 3D space. This 3D point $\mathbf{X}$ is projected back by the two camera matrices to the 2D points $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ on the camera image planes. In other words:

$$\begin{cases} \overrightarrow{\mathbf{x}^{(1)}} = \overrightarrow{\mathbf{P}^{(1)}\mathbf{X}} \\ \overrightarrow{\mathbf{x}^{(2)}} = \overrightarrow{\mathbf{P}^{(2)}\mathbf{X}} \end{cases} \tag{2.107}$$

These equations can be combined into a closed form $\mathbf{AX} = 0$, which is linear in the unknown elements of X. Consider the cross product of the vectors corresponding to each point in the image plane (like $\mathbf{x}^{(1)}$ or $\mathbf{x}^{(2)}$ in figure 2.35) with the vector corresponding to the point projected back by the corresponding camera matrix ($\overrightarrow{\mathbf{P}^{(1)}\mathbf{X}}$ or $\overrightarrow{\mathbf{P}^{(2)}\mathbf{X}}$ in figure 2.35). In an ideal case, this cross product should be zero, since all the projections of a point X in 3-space onto some camera image planes should correspond to a unique point on the ground plane. Therefore:

$$\left[\mathbf{x}^{(i)}\right]_x \hat{\mathbf{x}}^{(i)} = 0, i = 1,2,3 \tag{2.108}$$

in which:

$$\hat{\mathbf{x}}^{(i)} = \mathbf{P}^{(i)}\mathbf{X}, \qquad \mathbf{P}^{(i)}\text{: camera matrix for view i}$$

By joining the equations from 2 views, we get:

$$\begin{bmatrix} [\mathbf{x}^{(1)}]_x \mathbf{P}^{(1)} \\ ----- \\ [\mathbf{x}^{(2)}]_x \mathbf{P}^{(2)} \end{bmatrix} \mathbf{X} = \mathbf{AX} = 0 \tag{2.109}$$

Equation (2.109) will result in 6 linear equations, of which only 4 are linearly independent due to the homogeneous scale factor. Therefore, we have a total of 4 equations in 4 homogeneous unknowns in this case. Finally, one may be able to calculate the singular value decomposition for $\mathbf{A}$ and choose the right eigenvector corresponding to the smallest singular value of $\mathbf{A}$ as the optimal solution for $\mathbf{X}$.

3 Proposed Methodology

The previous part provided the basis for point tracking scheme and different methodologies for estimating geometric transformations. Our proposed methodology for estimation of visual object trajectory is summarized in the following block diagram.

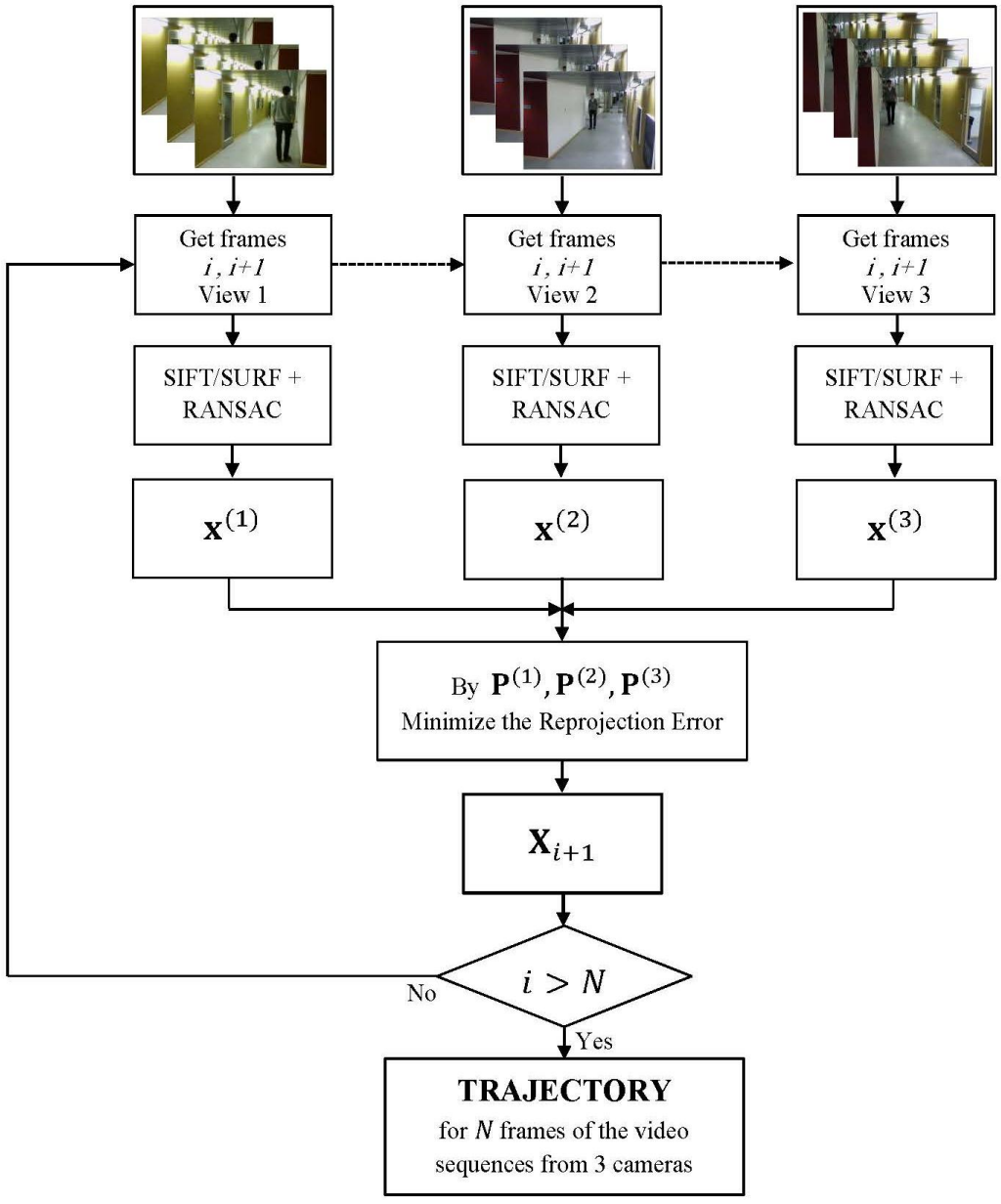


Figure 3.1: *Our methodology for estimation of visual object trajectory by using multiple cameras.*

As mentioned in the introduction, a significant part for any tracker is the initialization of the target. In this thesis, it is assumed that the target is present and fully visible in the starting frame of the video sequence. For this purpose, we manually select the area around the object in the first frame of all the views to generate the so call Region Of Interest or ROI.

Figure 3.2: *Examples of manual initialization of object in the 1st frame of video*

This ROI, is displayed in the video sequence as a coloured rectangle, and can be represented in vector or matrix form:

$$[x\ y\ width\ high] \quad ; \quad \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{bmatrix} \tag{3.1}$$

and in homogeneous coordinates system:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.2}$$

The aim of this chapter is to describe the methodology for tracking and deriving the trajectory of object movement on the ground plane with multiple cameras. The whole process is shown in figure 3.1.

Hence, this chapter is organized as follows. In the first part, we will discuss about how to estimate the best 2D transformation model given a set of point correspondences in two images. In the second part, the methodology for deriving the camera matrices as a 3D to 2D transformation will be described. Refer to section 2.2.6 for more details about the fundamentals of camera matrix. In the third part, we will go through the methodology for deriving the trajectory of object movement given the point correspondences and the camera matrices. Finally, in the fourth part of this

chapter, we will propose some covariance-based metric for occlusion which may be helpful for the tracker to recover the true position of the target in the occluded view.

3.1 2D Transformation Models Estimation

Here the focus will be on 2D transformations obtained by point correspondences between 2 frames of either a single view or multiple views. For transforming the object polygon among two frames, the number of candidate models will be directly related to the number of existing correct point correspondences. According to the requirement for algorithm 2.7 (normalized 8-point algorithm), 2 cases are mainly considered:

1. $n < 8$: 7 Simple 2D Transformation Models
2. $n \geq 8$: 7 Simple and 1Advanced 2D Transformation Model

Where *n* is the number of existing correct point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$.

For the 1st case, according to table 2.2, the candidate models (2D transformation models) will be only 7. On the other hand, for the 2nd case, according to algorithm 2.7, the fundamental matrix can be computed. So, in the 2nd case, we will have 8 candidate models to verify. These cases will be described in the next sections.

3.1.1 Simple 2D Models

In this section, the mathematics behind the estimation of 2D linear transformations is explained, especially for a 2D affine transformation. For other types of transformations, the process is quite the same except some careful considerations for each kind of transformation. Given 2 set of correspondent points $\mathbf{x}^{(i)}$, $\mathbf{x}^{(j)}$ the objective is to estimate the transformation **T** which satisfies the following equation:

$$\mathbf{x}^{(i)} \xrightarrow{\mathbf{T}} \mathbf{x}^{(j)} \quad or \quad \mathbf{x}^{(j)} = \mathbf{T}\mathbf{x}^{(i)} \tag{3.3}$$

Assume **T** is an affine transformation. So the general parametric form is given as:

$$\mathbf{T} = \begin{bmatrix} s\cos\theta & -\sin\theta & t_x \\ s\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

For 1 point pair, from equation (3.3) we would have:

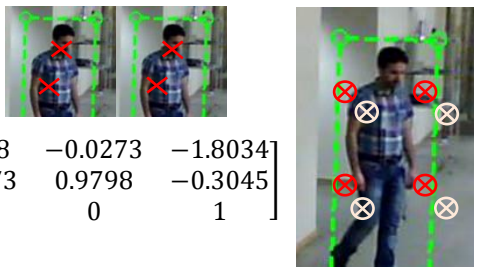
$$\begin{bmatrix} x^{(j)} \\ y^{(j)} \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{(i)} \\ y^{(i)} \\ 1 \end{bmatrix} \tag{3.5}$$

In order to reach the form of $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{x}$ and $\mathbf{b}$ are vectors and $\mathbf{x}$ is to be estimated by knowing $\mathbf{A}$, $\mathbf{b}$) after rearranging the equations in (3.5), we would have:

$$\begin{bmatrix} x^{(j)} \\ y^{(j)} \\ 1 \end{bmatrix} = \begin{bmatrix} x^{(i)} & y^{(i)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x^{(i)} & y^{(i)} & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

$$\Longrightarrow \mathbf{Ah} = \mathbf{b};\ \mathbf{A} = \begin{bmatrix} x^{(i)} & y^{(i)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x^{(i)} & y^{(i)} & 1 \end{bmatrix},\ \mathbf{h} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} x^{(j)} \\ y^{(j)} \\ 1 \end{bmatrix} \quad (3.6)$$

In this case, we have 6 unknowns in $\mathbf{h}$ which means at least 6 independent equations should be available. Also, by each point pair, 2 equations are derived (i.e. two rows in $\mathbf{A}$). Therefore, at least 3 point pairs are needed for an affine transformation to be estimated. The more the point pairs, the better estimation we get from solving equation (3.6). For solving (3.6), refer to section 2.2.5 in the applications of SVD method for solving linear system of equations. Practically, one must make sure the estimation results are accurate enough and the matrix $\mathbf{A}$ is well-conditioned (i.e. not close to singular). For this purpose, there is a function in MATLAB named as *cond* which is meant to be a measure for the accuracy of a solution to a system of linear equations. More specifically, this function can be used to assign a number to matrix $\mathbf{A}$. This number either shows the accuracy of a solution or the sensitivity of solution to errors in the data. A value close to 1 for this measure indicates a well-conditioned matrix. Mathematically, this function returns the 2-norm condition number, which is the ratio of the largest singular value of let say $\mathbf{A}$ to the smallest one. A simple visualization of the results for such an estimation is done in figure 3.3.



$$T = \begin{bmatrix} 0.9798 & -0.0273 & -1.8034 \\ 00.0273 & 0.9798 & -0.3045 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 3.3: *Example for estimation of a 2D transformation (bottom-left) given a set of point correspondences (Red crosses in the top-left images). In the right image, the red circles are transformed by T to obtain white circles which will contain the object area in the next frame.*

3.1.2 Excluding Outliers

According to section 2.1.5, for getting reliable results from the transformation estimation algorithm, the input point pairs must precisely match. Given n true point correspondences and k false point correspondences (either false negative or false positive), the set of point correspondences contains $n+k$ members. In figure 3.4, this problem is visualized for more clarity.

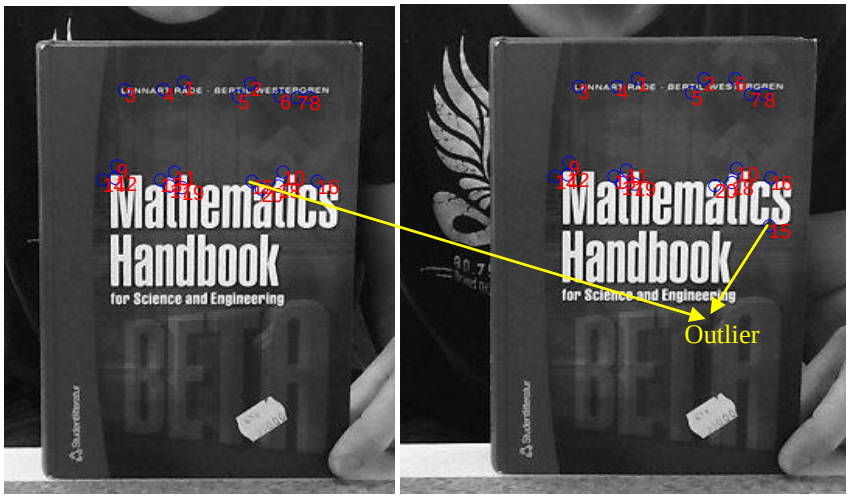


Figure 3.4: *Problem of outliers in the set of point correspondences*

As there are many ways to exclude outliers depending on the way they have been introduced, different kinds of outliers and the way of excluding them from the set of $n+k$ point matches are discussed in the following paragraphs.

Although the perfect matching for non-rigid objects and in case of uncorrelated camera views is not achieved, after excluding some outliers from the set of point correspondences by RANSAC or its variants, this set becomes much more accurate as an input for the transformation estimation algorithm (either simple or advanced models).

As described before, only from a sufficiently large set of n point correspondence, where $n \geq 8$, the estimated transformation can have all the interesting properties of the fundamental matrix $\mathbf{F}$. For special cases of motion the computation of the fundamental matrix is simplified. In all such cases, the number of degrees of freedom for the fundamental matrix will be less than 8.

According to our experiments, the fundamental matrix is much more sensitive to outliers and hence the problem of outliers is much more critical in case of the fundamental matrix computation.

There are basically two types of outliers:

- One is due to bad locations. In the estimation of the fundamental matrix, the location error of a point of interest is assumed to exhibit Gaussian behavior. This assumption is reasonable since the error in localization for most points of interest is small, but a few points are possibly incorrectly localized. The latter points will severely degrade the accuracy of the estimation. Due to the limited performance of a corner detector or low contrast of an image, a few points are possibly poorly localized. These outliers (sometimes even one) will severely affect the precision of the fundamental matrix if we directly apply the methods of SIFT/SURF without any post processing like RANSAC and its variants.
- The other possible reason to have outliers is the false matches. In fact when only heuristics metrics have been used to construct the set of point correspondences, the numbers of false matches would be higher. This will completely spoil the estimation process, and the final estimate of the fundamental matrix will be useless. For this reason, a separate distance measure is specified for inliers corresponding to the fundamental matrix based on the epipolar constraint in section 3.1.2.1.

A list of different methods for dealing with the problem of outliers is provided in Table 3.1. In this table, TE refers to section 3.1.1 for 2D transformation estimation.

Table 3.1: *Methods for estimating a transformation by RANSAC and its variants*

Method	Minimum Number of Points	Needed Assumptions
Transformation Estimation (TE)	1	Precisely matched point pairs (all the input points are inliers)
RANSAC + TE	1	The distance threshold for the inliers, (Desired Confidence q) or K
MSAC[1] + TE	1	The distance threshold for the inliers, (Desired Confidence q) or K (converges faster than RANSAC)
LMedS[2] + TE	2	*50%* of input point pairs as inliers, K
LTS[3] + TE	$\frac{100}{\%inliers}$	*% inliers* of input point pairs as inliers (converges faster than LMedS), K
RAMOSAC+TE	1	(Desired Confidence q) or K
BMSAC+TE	1	(Desired Confidence q) or K

1. M-estimator Sample Consensus.
2. Least Median of Squares.
3. Least Trimmed Squares.

The last method in table 3.1 will be described in section 3.1.3. Other methods iteratively calculate and compare a distance metric just the same as RANSAC algorithm (section 2.1.6) for the transformation model. Notice different assumptions used by each of these methods in table 3.1. This iterative process stops when either:

- *N* number of random sampling iterations is performed (should be explicitly specified for RAMOSAC. For other methods, it can be either explicitly specified or automatically obtained by a desired confidence rate).
- A certain number of inliers are found (dynamically updating *N*). This criterion only holds for RANSAC/MSAC.

The number of iteration *N* could be either explicitly specified or automatically computed based on some accuracy requirements. In the latter case, which is only

applicable to RANSAC and MSAC, the Desired Confidence (in percentage) level drives the accuracy and *N* is computed as:

$$N = \frac{\log(1-p)}{\log(1-q^s)} \tag{3.7}$$

where

- *p* is the probability of independent point pairs being an inlier, i.e. can be mapped by the same transformation. This probability represents the Desired Confidence and could be also dynamically calculated based on the number of inliers found versus the total number of points. As the probability increases, the number of samplings K decreases.
- *q* is the probability of finding the maximum number of inliers. *s* is equal to the value 2, 3, or 4 and 8 for similarity, affine, projective and 2D planar homography transformation, respectively. For other transformations, it is equal to 1.

In RANSAC and MSAC methods, *N* is updated according to the following rule in each iteration:

$$N = min(N, \frac{log(1-p)}{log(1-r^s)}) \tag{3.8}$$

where

$$r = \frac{\sum_{i=1}^{Num} sgn(D\left(p_i^b, \Psi\left(p_i^a : H\right)\right), t)}{Num} \tag{3.9}$$

The parameters used in these equations are described later in this section.

As could be seen in table 3.1, the main advantage of LMedS, LTS over RANSAC, MSAC is that they do not need a distance threshold to be explicitly specified. In algorithm 3.1, the transformation estimation by RANSAC and its variants is summarized.

Input:

- A set of point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$.
- A set of motion models (the possible models are in total 8).

Output:

- **T** 3x3 transformation matrix as the best transformation model.
- Inliers Set.

1. A transformation matrix $\mathbf{T}$, $\mathbf{T}_{\text{best}}$ is initialized to zeros
2. Set $\text{count} = 0$ (Randomly sampling iteration number).
3. While $\text{count} < k$, where k is total number of random sampling iterations to perform, do the following;
    a. Increment the count; $\text{count} = \text{count} + 1$.
    b. Randomly select pair of points from images **A** and **B**, (2 pairs for similarity, 3 pairs for affine, 4 pairs for projective, 8 points for fundamental matrix and planar homography).
    c. Calculate a transformation matrix **T**, from the selected points.
    d. If **T** has a distance metric less than that of $\mathbf{T}_{\text{best}}$, then replace $\mathbf{T}_{\text{best}}$ with **T**.

    (Optional for RANSAC algorithm only)

    i. Update k dynamically (based on some confidence rate, i.e. the probability of the algorithm to find the maximum number of inliers, following with the maximum number of iterations)
    ii. Exit out of sampling loop if enough number of point pairs can be mapped by $\mathbf{T}_{\text{best}}$ (Specified by a percentage of inliers manually).
4. Use all point pairs in images **A** and **B** that can be mapped by $\mathbf{T}_{\text{best}}$ to calculate a refined transformation matrix $\mathbf{T}'_{\text{best}}$
5. Iterative Refinement, (Optional for RANSAC and LMS algorithms)
    a. Denote all point pairs that can be mapped by $\mathbf{T}'_{\text{best}}$ as inliers (Find a new set of inliers).
    b. Use inliers' point pairs to calculate a transformation matrix **T**.
    c. If **T** has a distance metric less than that of $\mathbf{T}'_{\text{best}}$, then replace $\mathbf{T}'_{\text{best}}$ with **T**, otherwise exit the loop.

Algorithm 3.1. *Transformation Estimation with RANSAC/MSAC/LMedS/LTS*

3.1.2.1 Distance Measure for Inliers

For each of the transformations listed in table 2.2 and for the planar homography derived from the fundamental matrix (algorithm 2.8, 2.9), a distance measure is used to identify a point as outlier. This distance measure simply specifies the upper limit distance a point can differ from the projection location of its corresponding point. There are 2 alternatives for this measure:

- Algebraic Distance:

$$\begin{cases} d(\mathbf{a},\mathbf{b}) = (\mathbf{b}\mathbf{F}\mathbf{a}^T)^2 & : For\ Fundamental\ Matrix \\ d(\mathbf{a},\mathbf{b}) = (\mathbf{b}-\mathbf{T}\mathbf{a})^2 & : For\ other\ transformations \end{cases} \tag{3.10}$$

- Sampson Distance (Only applicable for fundamental matrix)

$$d(\mathbf{a},\mathbf{b}) = (\mathbf{bFa}^T)^2 \left[\frac{1}{(\mathbf{Fa}^T)_1^2+(\mathbf{Fa}^T)_2^2} + \frac{1}{(\mathbf{bF})_1^2+(\mathbf{bF})_2^2}\right] \quad (3.11)$$

Where $\mathbf{a}, \mathbf{b}$ are a matching point pair in two images and $(\mathbf{A})_j^2$ is the square of the j-th entry of the vector $\mathbf{A}$ and $\mathbf{F}$ is the fundamental matrix and $\mathbf{T}$ is a transformation matrix of those mentioned in table 2.2.

For now, algebraic distance is chosen since it is simpler and more efficient regarding the computational costs. More specifically, the algebraic distance for a pair of points, $[x^a \quad y^a]$ on image a, and $[x^b \quad y^b]$ on image b, according to transformation $\mathbf{T}$ is defined as follows.

For projective transformation:

$$D\left(\mathbf{x}_i^b, \Psi(\mathbf{x}_i^a : \mathbf{T})\right) = ((\hat{u}^a - \hat{w}^a x^b)^2 + (v^a - \hat{w}^a y^b)^2)^{\frac{1}{2}} \quad (3.12)$$

where

$$\begin{bmatrix}\hat{u}^a \\ \hat{v}^a \\ \hat{w}^a\end{bmatrix} = \mathbf{T}\begin{bmatrix}x^a \\ y^a \\ 1\end{bmatrix}$$

and for other transformations:

$$D\left(\mathbf{x}_i^b, \Psi(\mathbf{x}_i^a : \mathbf{T})\right) = ((\hat{x}^a - x^b)^2 + (\hat{y}^a - y^b)^2)^{\frac{1}{2}} \quad (3.13)$$

where

$$\begin{bmatrix}\hat{x}^a \\ \hat{y}^a \\ 1\end{bmatrix} = \mathbf{T}\begin{bmatrix}x^a \\ y^a \\ 1\end{bmatrix}$$

It is worth to mention that in case of having an ideal $\mathbf{T}$:

$$x^b = \frac{\hat{u}^a}{\hat{w}^a}, y^b = \frac{\hat{v}^a}{\hat{w}^a} \; or \; \begin{bmatrix}\hat{u}^a \\ \hat{v}^a \\ \hat{w}^a\end{bmatrix} \equiv \begin{bmatrix}x^b \\ y^b \\ 1\end{bmatrix} = \mathbf{T}\begin{bmatrix}x^a \\ y^a \\ 1\end{bmatrix} \quad (3.14)$$

Therefore, a pair of points, $\mathbf{x}_i^a$ (image a) and $\mathbf{x}_i^b$ (image b) is an inlier only when the distance between $\mathbf{x}_i^b$ and the projection of $\mathbf{x}_i^a$ by the transformation matrix $\mathbf{T}$ falls within the specified distance threshold.

- **Note**: This section does not apply to LMedS and LTS methods since they do not explicitly need a distance threshold for determining the inliers.

3.1.2.2 Fitness Metric for a Transformation

Finally, for assigning the transformation matrix **T** a score, a fitness metric is defined based on all the distances of the points and their projection location by this transformation matrix **T**, as mentioned earlier.

- The fitness metric for RANSAC: (The greater the value, the better the fitness)

$$d = \frac{\sum_{i=1}^{Num} \mathrm{sgn}(D\left(\mathbf{x}_i^b, \Psi(\mathbf{x}_i^a:\mathbf{T})\right), t)}{Num} \tag{3.15}$$

  where

$$sgn(a,b) = \begin{cases} 1 & a \le b \\ 0 & otherwise \end{cases}$$

- The fitness metric for LMedS: (The smaller value, the better the fitness)

$$d = median(D\left(\mathbf{x}_1^b, \Psi(\mathbf{x}_1^a:\mathbf{T})\right), \ldots, D\left(\mathbf{x}_{num}^b, \Psi(\mathbf{x}_{num}^a:\mathbf{T})\right)) \tag{3.16}$$

- The fitness metric for MSAC: (The smaller value, the better the fitness)

$$d = \sum_{i=1}^{Num} min(D\left(\mathbf{x}_i^b, \Psi(\mathbf{x}_i^a:\mathbf{T})\right), t) \tag{3.17}$$

- The fitness metric for LTS: (The smaller value, the better the fitness)

$$d = \sum_{i \in \Omega} D\left(\mathbf{x}_i^b, \Psi(\mathbf{x}_i^a:\mathbf{T})\right) \tag{3.18}$$

  where Ω is a set of point pairs corresponding to the first lower $(\%inliers * Num)$ of distances.

For all the fitness metric equations:

$\mathbf{x}_i^a$: a point in image *a*

$\mathbf{x}_i^b$: a point in image *b*

$\Psi(\mathbf{x}_i^a:\mathbf{T})$: the projection of a point on image *a* by a transformation matrix **T**

$D\left(\mathbf{x}_i^b, \mathbf{x}_j^b\right)$: the distance between two point pairs on image *b*

t : threshold

Num: Number of point pairs

3.1.3 Best Model and SAmple Consensus

Here, we propose another variant of RAMOSAC and bring up with its advantages and disadvantages over other variants. We also name this method as BMSAC. The basic idea is to do a full search through all the candidate models used in RANSAC and choose the one with the highest score. So, the main difference of BMSAC with RAMOSAC will be the fact that in RAMOSAC the best model is found in a random and iterative process of model selection and scoring while in BMSAC all the models are verified and assigned a score. Hence, the steps in BMSAC algorithm will be quite the same as RAMOSAC. The point-based features tracking scheme with multiple models using BMSAC is summarized in figure 3.5.

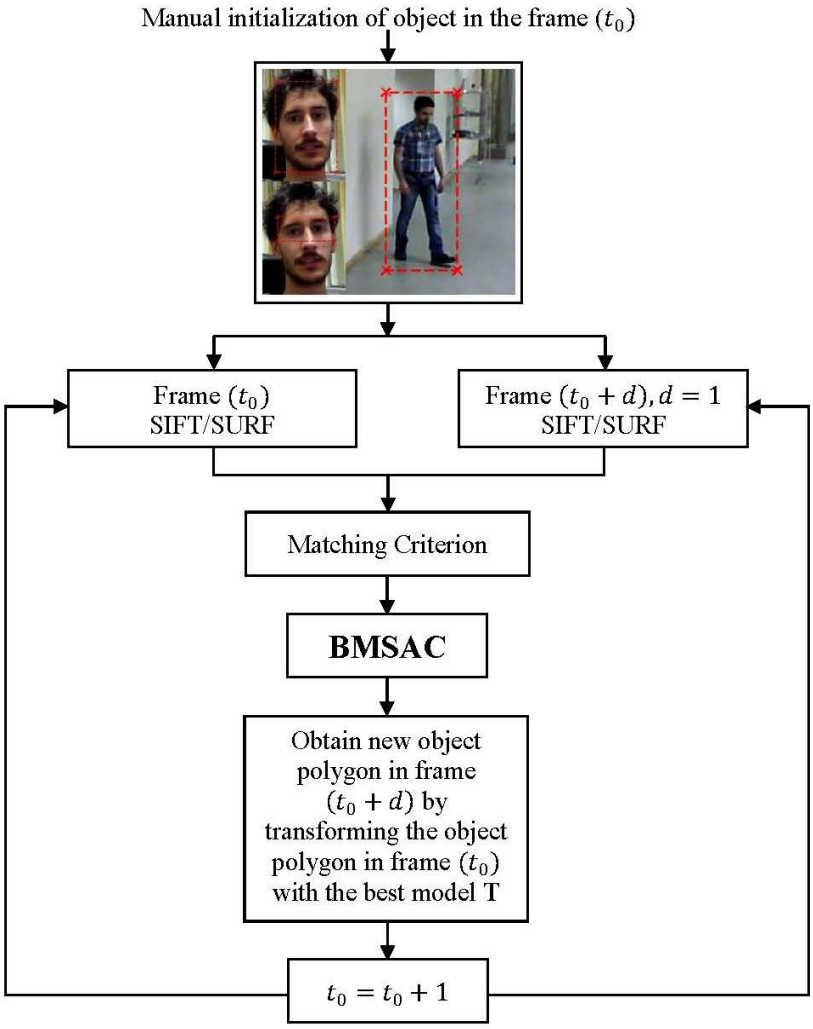


Figure 3.5: *The point-based features tracking scheme with multiple models using BMSAC*

The advantage of BMSAC over RAMOSAC is the guarantee for the best model among the candidates to be found and the best set of inliers (point correspondences). However, as this method needs a full search through all the model candidates, it will be more computationally intensive than RAMOSAC. Therefore, for real-time applications, RAMOSAC will be a better choice. The steps for the proposed method are summarized in algorithm 3.2.

Input:

- Set of point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$ from two images.
- Target bounding box expressed in homogeneous coordinates in the previous frame.
- Weighted mean of previous samples for an exponential distribution.
- Threshold for RANSAC.

Output:

- 3x3 matrix **T**.
- Inliers.
- Updated weighted mean;

1. According to the number of point correspondences given as input, decide how many models could be evaluated as below:

N<8: *numModels* = 6 (first 6 models)
N<3: *numModels* = 5 (first 5 models)
N<2: *numModels* = 3 (first 3 models)

2. For *modelType* : 1 to *numModels* do

a. Find the best set (with the largest number of inliers) of inliers consisting with *modelType* by RANSAC.
b. Estimate **H** (the model) by only the best set of inliers calculated in a.
c. Compute the distance which this transformation imposes on the previous box. (simply Euclidean distance of the corner points for two bounding box which are meant to be transformed to each other)
d. Get the weighted mean of the probability density function for this transformation by the following formula:
*weighted_mean = delta*prev_weighted_mean + (1-delta)*distance*
(Assuming that the distance follows some types similar to exponential function which is proved to be suitable by plotting the distribution for this distance in different scenarios)
e. The value for *weighted_mean* in (d) is the same as $1/\lambda$ for an exponential function. So the parameter λ is known and the PDF of this transformation would be:

$$prob = e^{-\lambda * distance}$$

f. Assign a score to this transformation based on the following formula (the same as scoring scheme for RAMOSAC is used):

$$\text{score} = \#\text{inliers} + \log10(\text{prob}) + \varepsilon * \text{nmin}$$

3. Choose the model with the highest score.

Algorithm 3.2. *BMSAC*

3.2 Computation of Camera Matrices

In this section, a method is proposed for the computation of camera projection matrices given the fundamental matrices **F** and point correspondences $\mathbf{x}^{(i)} \leftrightarrow \mathbf{x}^{(j)}$. In other words, by using the epipolar geometry and without calibrating the cameras, we would be able to compute the camera matrices by some reconstruction ambiguity in 3-space. In figure 3.6, the proposed methodology is summarized.

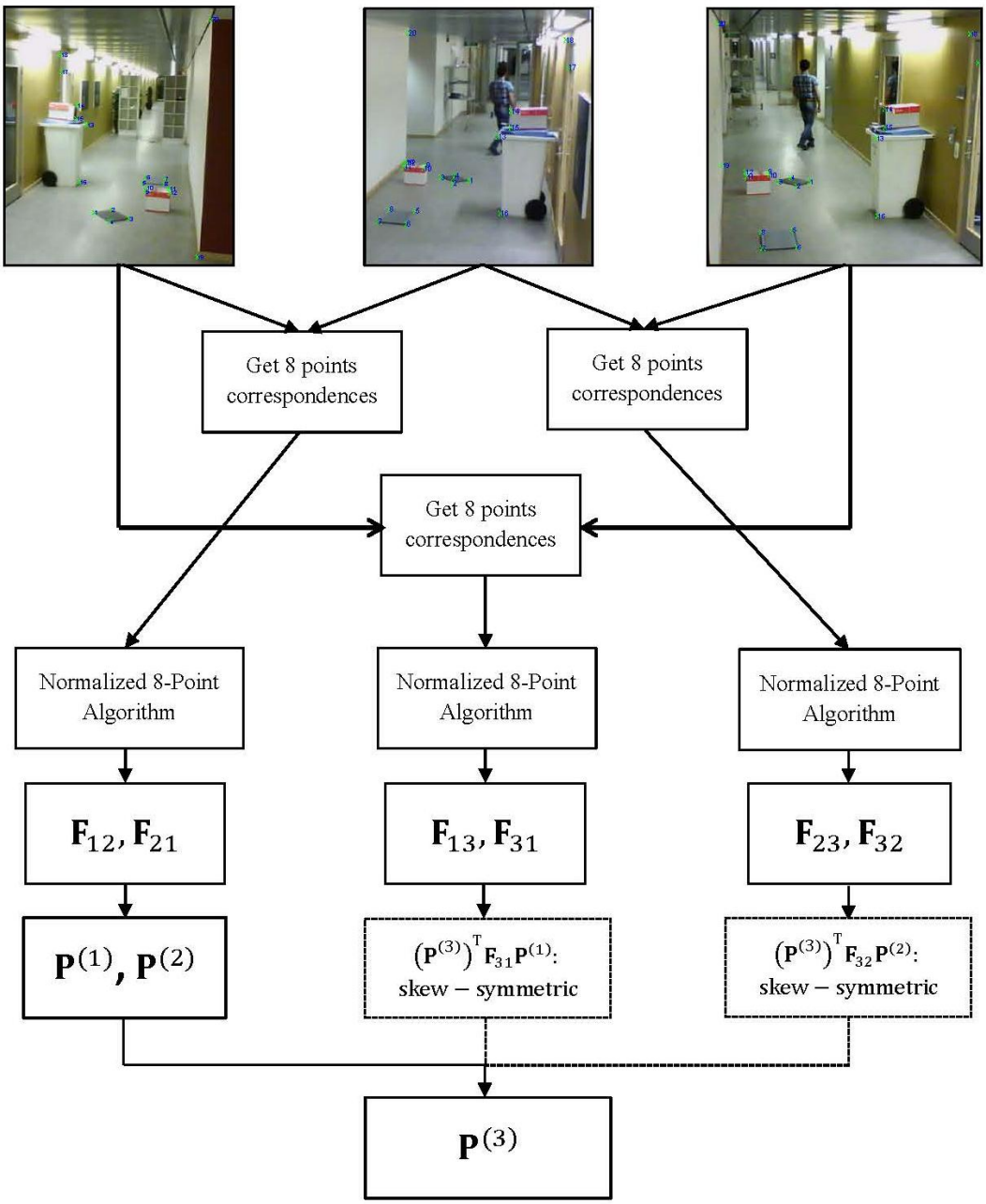


Figure 3.6: *Our methodology for derivation of camera matrices by using Epipolar Geometry*

This reconstruction ambiguity is expressed by an arbitrary *4x4* invertible projective transformation $\mathbf{H}$ and it means by replacing points $\mathbf{X}_i$ by $\mathbf{H}\mathbf{X}_i$ and matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ by $\mathbf{P}^{(1)}\mathbf{H}^{-1}$ and $\mathbf{P}^{(2)}\mathbf{H}^{-1}$ the image point coordinates do not change. This shows that the points $\mathbf{X}_i$ and the camera matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ can be determined at best only up to a projective transformation. This is the only ambiguity in the reconstruction of points from two images. Thus reconstruction from uncalibrated cameras is possible up to a projective transformation [1]. In the following figure, this ambiguity of reconstruction is visualized.

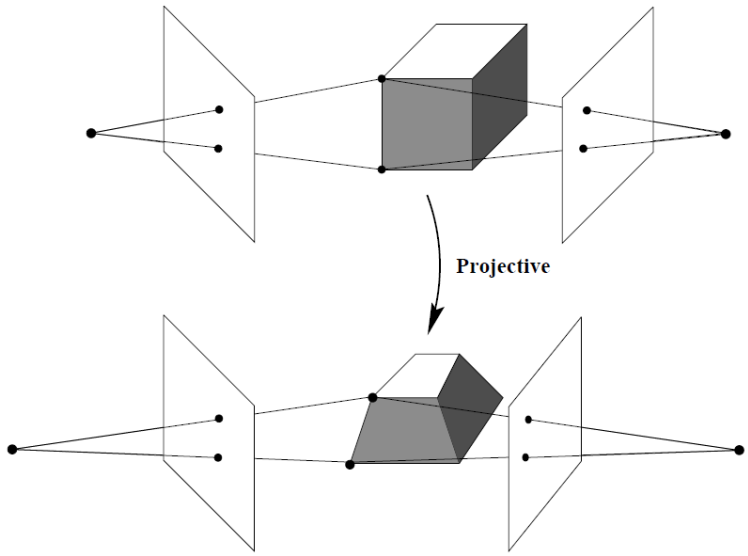


Figure 3.7: *If the cameras are uncalibrated then reconstructions only lead to the image points (the intersection of the rays with the image plane). A projective transformation of the structure and camera positions does not change the measured points, although the angle between rays is altered. The epipoles are also unchanged (intersection with baseline). This figure is taken from [1].*

Note: For the set of point correspondences needed for the computation of **F**, we may have two different options:

1. Manually specify at least 8 points belonging to the static objects of the scene that are visible in all the views.
2. Automatically detect and match at least 8 points among each frame of different views.

If the scene is not much different regarding the appearance of objects in the scene from different views, by the second way, a set of point correspondences may be determined by SIFT/SURF (algorithm 2.1, 2.2). This way is especially suitable when a camera is moving and one looks for the trajectory of camera movement along the

video frames. Otherwise, which is the case for our thesis, we choose the first way to make a synthetic set of point correspondences needed for the calculation of fundamental matrices **F**. In other words, the views in our case are disjoint which means the cameras are much different regarding their view angle of the scene and position in the 3D world coordinate system. In figure 3.8, 20 point correspondences among 3 views in one of our scenarios is shown, which is used to derive the fundamental matrices **F**.

Figure 3.8: *20 point correspondences among 3 disjoint views*

An important assumption in this thesis is that both the object and the cameras have movement. This means that the camera parameters are assumed to be unknown. This means both the scene and the cameras uncalibrated and also if the camera moves,

there needs to be some intelligent methods that are able to find a correct set of point correspondences between disjoint views of the scene (as in figure 3.8).

According to result 9.15 of [1], we can derive a general formula for a pair of canonical camera matrices corresponding to a fundamental matrix **F** that is:

$$\mathbf{P}^{(1)} = [\mathbf{I}|0], \quad \mathbf{P}^{(2)} = \left[\left[\mathbf{e}^{(2)}\right]_x \mathbf{F}_{12} \mid \mathbf{e}^{(2)}\right] \tag{3.20}$$

where $\mathbf{e}^{(2)}$ is the epipole in the view corresponding to camera matrix $\mathbf{P}^{(2)}$. The notation $\left[\mathbf{e}^{(2)}\right]_x$ is the skew-symmetric matrix corresponding to $\mathbf{e}^{(2)}$ defined in equation (2.48). The first part of equation (3.20) implies that the origin of world coordinate system is fixed at the centre of the first camera. In addition, all the cameras obey from the pin-hole camera model.

In order to understand the reason for choosing such general form for $\mathbf{P}^{(2)}$, we refer again to equation 2.42 for the algebraic representation of epipolar constraint by the fundamental matrix **F**:

$$(\mathbf{x}^{(2)})^T \mathbf{F}_{12}\, \mathbf{x}^{(1)} = 0 \tag{3.21}$$

$$\mathbf{x}^{(1)} = \mathbf{P}^{(1)}\mathbf{X}, \;\; \mathbf{x}^{(2)} = \mathbf{P}^{(2)}\mathbf{X} \tag{3.22}$$

By replacing the equations 3.22 in equation 3.21, we get:

$$\Rightarrow \mathbf{X}^{\mathrm{T}}(\mathbf{P}^{(2)})^{\mathrm{T}}\mathbf{F}_{12}\mathbf{P}^{(1)}\mathbf{X} = 0 \tag{3.23}$$

Since equation 3.23 holds for all **X** and according to the interesting properties of skew-symmetric matrices and having $\mathbf{P}^{(1)} = [\mathbf{I}|0]$, we must have:

$$(\mathbf{P}^{(2)})^{\mathrm{T}}\mathbf{F}_{12}\mathbf{P}^{(1)}\text{: Skew-Symmetric} \tag{3.24}$$

Right now, by some intuition from mathematics and skew-symmetric matrices, the following form is proposed for $\mathbf{P}^{(2)}$:

$$\mathbf{P}^{(2)} = \left[\mathbf{S}\mathbf{F}_{12}\middle|\mathbf{e}^{(2)}\right] \tag{3.25}$$

where **S** is any 3x3 skew-symmetric matrix. In order to prove this claim, one can write:

$$(\mathbf{P}^{(2)})^{\mathrm{T}}\mathbf{F}_{12}\mathbf{P}^{(1)} = \left[\mathbf{S}\mathbf{F}_{12}\middle|\mathbf{e}^{(2)}\right]^{\mathrm{T}}\mathbf{F}_{12}[\mathbf{I}|0] = \begin{bmatrix} (\mathbf{F}_{12})^{\mathrm{T}}\mathbf{S}^{\mathrm{T}}\mathbf{F}_{12} & \mathbf{0} \\ (\mathbf{e}^{(2)})^{\mathrm{T}}\mathbf{F}_{12} & 0 \end{bmatrix} = \begin{bmatrix} (\mathbf{F}_{12})^{\mathrm{T}}\mathbf{S}^{\mathrm{T}}\mathbf{F}_{12} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} \tag{3.26}$$

In which $(\mathbf{F}_{12})^{\mathrm{T}}\mathbf{S}^{\mathrm{T}}\mathbf{F}_{12}$ is again an skew-symmetric matrix.

By extending this result to the case for three cameras, the camera projection matrices for the three views are expressed as follow:

$$\mathbf{P}^{(1)} = [\mathrm{I}|0] \qquad \mathbf{P}^{(2)} = [\mathbf{A}|\mathbf{a}_4] \qquad \mathbf{P}^{(3)} = [\mathbf{B}|\mathbf{b}_4] \tag{3.27}$$

where:

- the first camera centre $C^{(1)}$ is assumed to be in the centre of the world coordinate system.
- **A** and **B** are *3×3* matrices (the infinite homography from the first to the second and third cameras respectively [1])
- $\mathbf{a}_i$ and $\mathbf{b}_i$ are vectors composed from the i-th columns of the respective camera matrices for i=1,…,4. In this case, $\mathbf{a}_4$ and $\mathbf{b}_4$ are the epipoles in view two and three respectively, arising from the first camera. These epipole will be denoted by $\mathbf{e}^{(2)}$ and $\mathbf{e}^{(3)}$ for the rest of this chapter.

There are for sure some geometric properties that are invariant under 3D projective transforms such as image coordinates and 3D incidence relations. So we are free to choose the camera in this form and certainly we capture the 3D structure of the scene by deriving the camera matrices in this way.

The first two camera matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ could be determined from the fundamental matrix $\mathbf{F}_{12}$ according to equation (3.20). We only need to determine the third camera matrix $\mathbf{P}^{(3)}$. In short, we cannot go through the same process for deriving the third camera matrix $\mathbf{P}^{(3)}$ since it will not be essentially compatible with projective ambiguity of $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$. Hence, the following steps are proposed for deriving the third camera matrix in order to make sure $\mathbf{P}^{(3)}$ will be in the same projective frame as $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$.

(i) Select a set of matching points $\mathbf{x}^{(1)} \leftrightarrow \mathbf{x}^{(2)}$ in the first two images, satisfying $(\mathbf{x}^{(2)})^T\mathbf{F}_{21}\mathbf{x}^{(1)} = 0$, and use triangulation to determine the corresponding 3D points **X** according to section 2.2.8.

(ii) Solve for the camera matrix $\mathbf{P}^{(3)}$ from the set of 3D-2D correspondences $\mathbf{X} \leftrightarrow \mathbf{x}^{(3)}$.

For the second step, we have the following equation to be solved:

$$\mathbf{x}^{(3)} = \mathbf{P}^{(3)}\mathbf{X} \tag{3.28}$$

$$\begin{bmatrix} x^{(3)} \\ y^{(3)} \\ z^{(3)} \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.29}$$

In order to solve the equation above, we use the result 9.12 of [1] which says:

A non-zero matrix **F** *is the fundamental matrix corresponding to a pair of camera matrices* $\mathbf{P}^{(1)}$ *and* $\mathbf{P}^{(2)}$ *if and only if* $(\mathbf{P}^{(2)})^T\mathbf{F}\mathbf{P}^{(1)}$ *is skew-symmetric.*

Therefore, the third camera matrix $\mathbf{P}^{(3)}$ must satisfy two conditions that is:

$$\begin{cases} (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{31}\mathbf{P}^{(1)}\text{: skew} - \text{symmetric} \\ (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{32}\mathbf{P}^{(2)}\text{: skew} - \text{symmetric} \end{cases} \tag{3.30}$$

Each of these conditions gives rise to 10 linear equations in the entries of $\mathbf{P}^{(3)}$, i.e. a total of 20 equations in the 12 entries of $\mathbf{P}^{(3)}$. From these, $\mathbf{P}^{(3)}$ may be computed linearly. For more clarity, a general form of a skew-symmetric matrix is shown with its element properties in the following equation:

$$\mathbf{A} = \begin{pmatrix} a_{11} = 0 & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} = 0 \end{pmatrix} \tag{3.31}$$

The matrix **A** is skew-symmetric if $a_{ij} = -a_{ji}$ for all $1 \le i \le n, 1 \le j \le n$. The main diagonal entries are zero because $a_{ii} = -a_{ii}$ implies $a_{ii} = 0$.

So, by knowing this symmetric property of two matrices $(\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{31}\mathbf{P}^{(1)}$, $(\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{32}\mathbf{P}^{(2)}$, we may rewrite them as follows:

$$\begin{aligned} \mathbf{B}_1 = (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{31}\mathbf{P}^{(1)} = (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{A}_1 &= \begin{bmatrix} p_1 & p_5 & p_9 \\ p_2 & p_6 & p_{10} \\ p_3 & p_7 & p_{11} \\ p_4 & p_8 & p_{12} \end{bmatrix} \begin{bmatrix} a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \\ a_3 & a_6 & a_9 & a_{12} \end{bmatrix} \\ &= \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \end{aligned} \tag{3.32}$$

$$\mathbf{B}_2 = (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{F}_{32}\mathbf{P}^{(2)} = (\mathbf{P}^{(3)})^{\mathrm{T}}\mathbf{A}_2 = \begin{bmatrix} p_1 & p_5 & p_9 \\ p_2 & p_6 & p_{10} \\ p_3 & p_7 & p_{11} \\ p_4 & p_8 & p_{12} \end{bmatrix} \begin{bmatrix} a'_1 & a'_4 & a'_7 & a'_{10} \\ a'_2 & a'_5 & a'_8 & a'_{11} \\ a'_3 & a'_6 & a'_9 & a'_{12} \end{bmatrix}$$

$$= \begin{bmatrix} b'_{11} & b'_{12} & b'_{13} & b'_{14} \\ b'_{21} & b'_{22} & b'_{23} & b'_{24} \\ b'_{31} & b'_{32} & b'_{33} & b'_{34} \\ b'_{41} & b'_{42} & b'_{43} & b'_{44} \end{bmatrix} \tag{3.33}$$

Then, by enforcing the following two properties of skew-symmetric matrices to $\mathbf{B}_1$ and $\mathbf{B}_2$, we get:

$$\begin{cases} b_{ij} = -b_{ji}, b'_{ij} = -b'_{ji} \Rightarrow b_{ij} + b_{ji} = 0, b'_{ij} + b'_{ji} = 0 \\ b_{ii} = 0, b'_{ii} = 0 \end{cases} \tag{3.34}$$

For all $1 \leq i \leq n, 1 \leq j \leq n$.

By rearranging the set of equations derived from previous step in the unknown elements of $\mathbf{P}^{(3)}$ and joining the results from $\mathbf{B}_1$ and $\mathbf{B}_2$, we reach the following set of 20 linear equations in the unknown elements of $\mathbf{P}^{(3)}$:

$$\mathbf{A}\mathbf{P}^{(3)} = 0 \tag{3.35}$$

where:

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 & 0 & 0 & a_2 & 0 & 0 & 0 & a_3 & 0 & 0 & 0 \\ 0 & a_4 & 0 & 0 & 0 & a_5 & 0 & 0 & 0 & a_6 & 0 & 0 \\ 0 & 0 & a_7 & 0 & 0 & 0 & a_8 & 0 & 0 & 0 & a_9 & 0 \\ 0 & 0 & 0 & a_{10} & 0 & 0 & 0 & a_{11} & 0 & 0 & 0 & a_{12} \\ a'_1 & 0 & 0 & 0 & a'_2 & 0 & 0 & 0 & a'_3 & 0 & 0 & 0 \\ 0 & a'_4 & 0 & 0 & 0 & a'_5 & 0 & 0 & 0 & a_{\prime 6} & 0 & 0 \\ 0 & 0 & a'_7 & 0 & 0 & 0 & a'_8 & 0 & 0 & 0 & a'_9 & 0 \\ 0 & 0 & 0 & a'_{10} & 0 & 0 & 0 & a'_{11} & 0 & 0 & 0 & a'_{12} \\ a_4 & a_1 & 0 & 0 & a_5 & a_2 & 0 & 0 & a_6 & a_3 & 0 & 0 \\ a_7 & 0 & a_1 & 0 & a_8 & 0 & a_2 & 0 & a_9 & 0 & a_3 & 0 \\ a_{10} & 0 & 0 & a_1 & a_{11} & 0 & 0 & a_2 & a_{12} & 0 & 0 & a_3 \\ 0 & a_7 & a_4 & 0 & 0 & a_8 & a_5 & 0 & 0 & a_9 & a_6 & 0 \\ 0 & a_{10} & 0 & a_4 & 0 & a_{11} & 0 & a_5 & 0 & a_{12} & 0 & a_6 \\ 0 & 0 & a_{10} & a_7 & 0 & 0 & a_{11} & a_8 & 0 & 0 & a_{12} & a_9 \\ a'_4 & a'_1 & 0 & 0 & a'_5 & a'_2 & 0 & 0 & a'_6 & a'_3 & 0 & 0 \\ a'_7 & 0 & a'_1 & 0 & a'_8 & 0 & a'_2 & 0 & a'_9 & 0 & a'_3 & 0 \\ a'_{10} & 0 & 0 & a'_1 & a'_{11} & 0 & 0 & a'_2 & a'_{12} & 0 & 0 & a'_3 \\ 0 & a'_7 & a'_4 & 0 & 0 & a'_8 & a'_5 & 0 & 0 & a'_9 & a'_6 & 0 \\ 0 & a'_{10} & 0 & a'_4 & 0 & a'_{11} & 0 & a'_5 & 0 & a'_{12} & 0 & a'_6 \\ 0 & 0 & a'_{10} & a'_7 & 0 & 0 & a'_{11} & a'_8 & 0 & 0 & a'_{12} & a'_9 \end{bmatrix}$$

$$\mathbf{P}^{(3)} = \begin{bmatrix} p_1 \\ \vdots \\ p_{12} \end{bmatrix}$$

For solving this homogenous system of linear equations, we look for $\mathbf{P}^{(3)}$ which minimizes $\mathbf{AP}^{(3)}$. This is a minimization problem which could be solved by least-squared minimization method. Also, one can use SVD method to choose the eigenvector corresponding to smallest eigenvalue of **A**. The second way is the same as least-square method. Finally, we may reshape $\mathbf{P}^{(3)}$ to get a *3×4* projection matrix corresponding to the third camera.

3.3 Derivation of Trajectory

In the calibrated case, a 3D structure can be recovered from two images only up to a rigid transformation and an unknown scale factor (this transformation is also known as a similarity), because we can fix the world coordinate system at an arbitrary point. Usually one chooses this point is chosen as the centre of one of the cameras. On the other hand, in the uncalibrated case, a 3D structure can only be recovered up to a projective transformation of the 3-space. In the uncalibrated case, like the one considered here, we assume that the fundamental matrix between the two images is known. Such case is known as weakly calibrated.

Given the camera matrices (computed in section 3.2) and 3 projections of a point **X** in 3D world coordinate system on the camera image planes ($\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}$), we may be able to calculate **X** by minimizing the algebraic distance as follow [1].

1) The first step is to normalize the point coordinates in the image plane of all the views according to the image sizes. The same process should be done for the camera matrices as well. These two tasks are named as preconditioning. The whole process of preconditioning can be considered as a two-stage process of scaling the coordinates down from *[0…image_width]* to *[0…2]* and translation from *[0…2]* to *[-1…1]*.
2) In the second step, we consider the cross product of each point in the image plane with its corresponding point on the ground plane (projected back by the corresponding camera matrix). In an ideal case, this cross product for all the points should be zero, since all the image points should correspond to a unique point on the ground plane. Therefore, we have the following set of equations:

$$\left[\mathbf{x}^{(i)}\right]_x \hat{\mathbf{x}}^{(i)} = 0, i = 1,2,3 \tag{3.36}$$

in which:

$$\hat{\mathbf{x}}^{(i)} = \mathbf{P}^{(i)}\mathbf{X}, \qquad \mathbf{P}^{(i)}\text{: camera matrix for view i}$$

By joining the equations from 3 views, we get:

$$\begin{bmatrix} [\mathbf{x}^{(1)}]_x \mathbf{P}^{(1)} \\ - - - - - \\ [\mathbf{x}^{(2)}]_x \mathbf{P}^{(2)} \\ - - - - - \\ [\mathbf{x}^{(3)}]_x \mathbf{P}^{(3)} \end{bmatrix} \mathbf{X} = \mathbf{A}\mathbf{X} = 0 \tag{3.37}$$

So, we look for a solution to this equation which minimizes **AX**. This solution can be easily obtained by decomposing **A** with singular value decomposition method and choosing the right eigenvector corresponding to the lowest singular value of **A**. This **X** could be given to Newton Iteration Estimation method as the initial estimate to get a more precise estimate of the true **X**.

This solution relies on the fact that the object is visible and being tracked so that the centre point of its image in all views could be determined. If the object is occluded in one or two views, it is probable that this method gives rise to wrong estimate for the true position of the object in the 3D world coordinate system (i.e., **X**).

One idea is to define some measure of occlusion for the object in each single view to identify how much the object is occluded. Then, by this measure we may be able to recognize if the occlusion has happened in some views or not. Afterwards, we may be able to recover the true position of object in the occluded view by the following steps to help the tracking system not to lose the object track in the occluded view:

1. Project back the object position from other views to the 3D world coordinate system by the corresponding camera matrices and triangulation.
2. Project the recovered object position in the 3D world coordinate system onto the occluded view by its camera matrix.

In addition, by such metric the algorithm will be able to understand if the occluded object becomes visible again. In the next section, some investigations will be made for such measure. Finally, in chapter 4, some results are shown for the performance of this measure on a couple of scenarios.

3.4 Occlusion Metric

In the last part of this chapter, we would like to open up the idea for defining a metric for occlusion that can hopefully be useful in both single and multi-view scenarios. Such metric should define a measure of visibility for the object in each frame by assuming a predefined appearance model confirming a certain percentage of visibility in the starting/reference frame.

According to such predefined appearance model, in the subsequent frames a similarity measure (quite robust towards illumination/appearance change) is used to assign a measure of occlusion for the object compared to the starting/reference frame.

It is preferred to have a mathematical function which maps the values for this similarity measure to the values in the range of 0-1 or 0-100. In such case, one would be able to decide how much to trust on the tracking of an object or not.

One good candidate for the appearance model could be the Covariance Matrix of some features calculated from the object. The covariance method is mathematically proved to be robust enough towards appearance and illumination changes. Refer to section 2.1.9 for more details. However, based on our experiments, the covariance-based appearance model may only be suitable for non-rigid objects [23].

After calculating the covariance matrix of the ROI, a relationship must be established as the similarity of covariance matrices for different ROIs and eventually defining the rate of occlusion. One possible measure is mentioned in section 2.1.9, equation 2.25 (Log-Euclidean Similarity Measure).

In order for this metric to be in the range of 0-1 or 0-100, by getting some implications from the log-euclidean similarity measure, we tried the following steps. For a set of images, the area of the occluded part of the object is computed by MATLAB function *polyarea*. Then, for each of the object ROIs, the covariance matrix and the similarity metric are computed, an example is shown in figure 3.9. The similarity measure for the two covariance matrices could also be obtained from the MATLAB function *corr2*. Based on our experiments, this similarity measure performs better than the log-euclidean similarity measure. In figure 3.10, the nonlinear relationship between the occlusion rate and similarity measure of covariance matrices is plotted.

Finally, by RANSAC or Least Square method, some polynomial and logarithmic functions are fitted onto these data points to estimate the suitable mathematical model that maps the similarity of the covariance matrices to the rate of occlusion which is in the range of 0-100. The result is shown in figure 3.11.

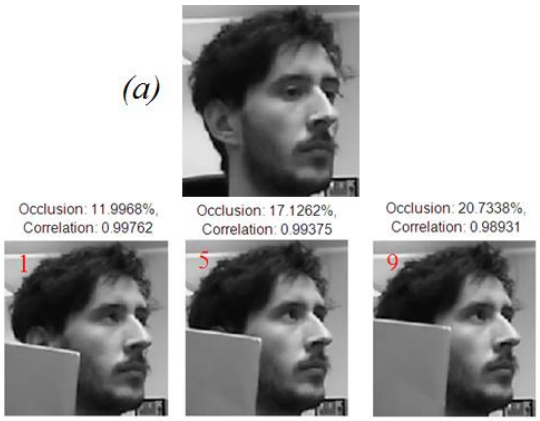


Figure 3.9: *Reference face image* ***(a)*** *and the rate of occlusion of its 3 sample occluded versions.*

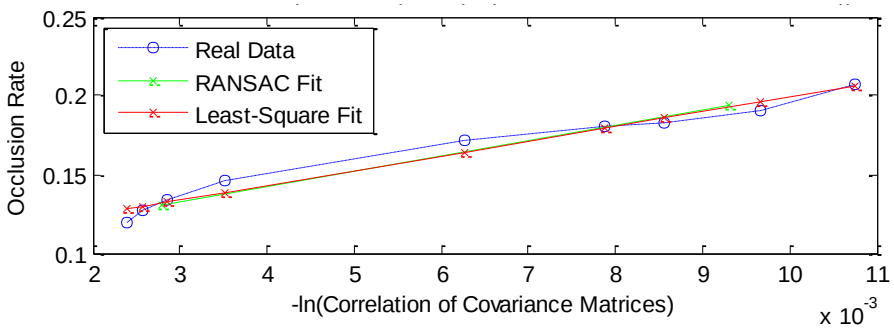


Figure 3.10: *Relation between (Occlusion Rate) and the natural logarithm of (Correlation of Covariance Matrices)*

In order to investigate the linearity, injectivity and slope of different logarithmic functions, 10 different functions are plotted. The domains in which these functions are plotted have been derived by enforcing two conditions on the function value:

$$\begin{cases} -\log(x^i) \geq 0 \\ -\log(x^i) \leq 100 \end{cases} \tag{3.38}$$

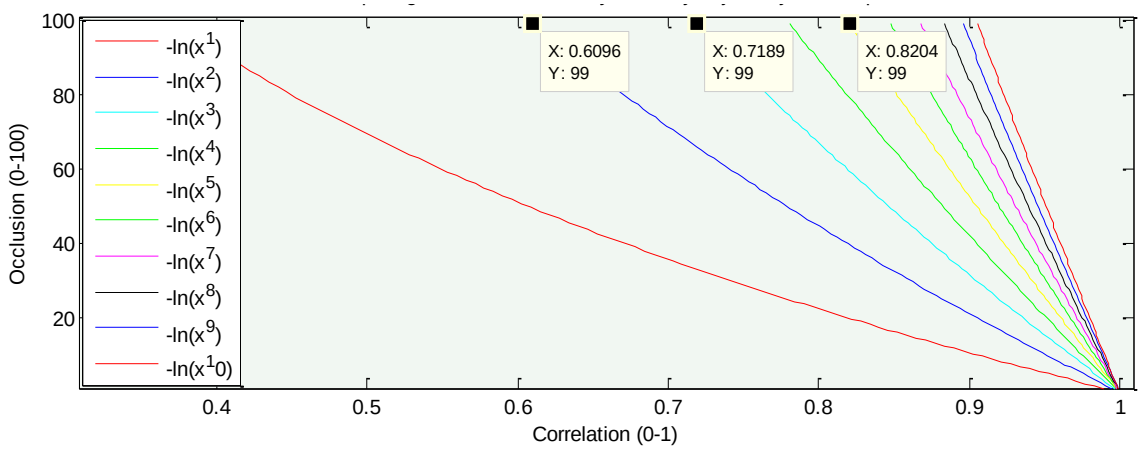


Figure 3.11: *Relation between (Occlusion Rates) and (Correlation of Covariance Matrices)*

As could be seen in this graph, as the exponent for *x* increases, the function shows more linear behaviour in the specified range that is visible in the figure. On the other hand, if the function is very steep, it could be very sensitive and for 0.001 changes in correlation, the occlusion rate would be changed more than 10 percent which is not suitable. So based on our evaluations, the third function from the left in this graph ($-ln(x^3)$) gives reasonable results in terms of good sensitivity and also linearity of the function.

Finally, for validating the derived formula for the occlusion rate, we measured its value in a self-made video of a human face from 3 views which is occluded by a plain paper gradually during the video in some of the views. In this scenario, 3 cameras were placed at about 50 cm distance from each other and all capturing the same scene. Then the object (a face) is tried to be occluded in one view while it is still visible in the other 2 views. The results in these curves clearly show that in most of the time durations, there is a dominant view in terms of object visibility which is absolutely correct. This could be another verification of the metric. Yet, there is still doubt if the same metric gives convincing results on different kind of scenarios and objects without precise adjustments of the two conditions (I,II) (which is very critical for this metric to work fine). The results of these experiments is summarized in figure 3.12.

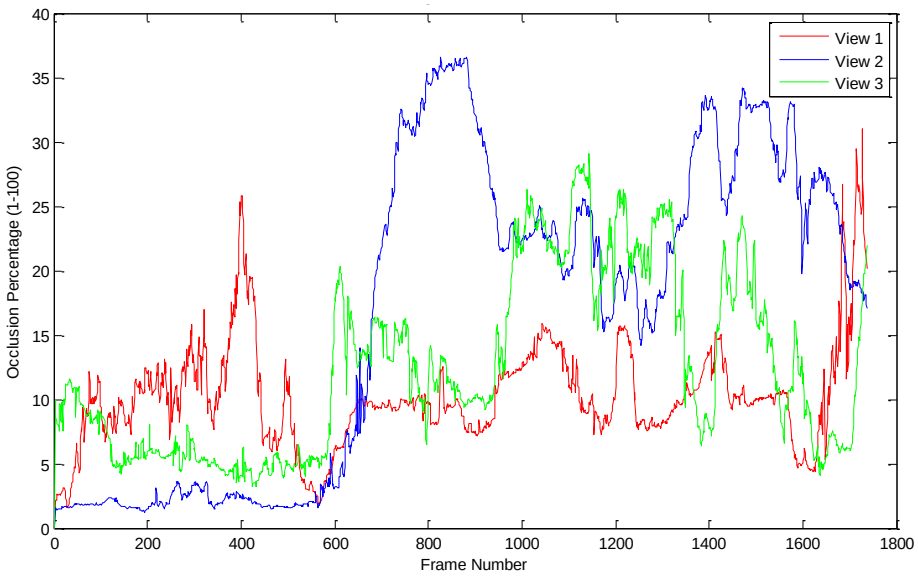


Figure 3.12: *Occlusion Rates versus #Frame in 3 views*

One possible way to improve the result by this method may be considering the covariance matrix from different views. In our case in fact, from each view one covariance matrix can be extracted, from the ROIs. If we save all these covariance matrices, we would have three reference matrices as a comparison. In this way, small changing of pose or appearance can be excluded from the case of occlusion.

Such definition and derivation of occlusion metric is considered as a trial based on an intuition for detecting occlusion rate of objects, and not based on strong mathematics. In simple words, if the target pose or appearance changes or the occluding object is much similar to the target itself, such covariance based metric will not work very well.

In short, if there is an intelligent way to learn the appearance of target by having its different views, one may rely on this knowledge to define a metric for occlusion. This is to be investigated more in future. In the following, a couple of parameters important for implementation of the covariance method are described and some information is given about how to correctly adjust them according to the specific scenario.

'imsize': An integer number specifying the size of image to be partitioned for extracting the covariance matrix of features from a target. Since covariance matrix is a computationally heavy process for large size targets, as their size is decreased, our experiments showed that the run time speed increased up to 5 times faster. The default value for this parameter is set to 32. But since this resized image of object is later partitioned, this parameter can be set to even higher values like 64 or 128.

'blocksize': An integer number (preferably of the power of 2 for efficient convolution operation) for the size of blocks in a partitioned version of resized object. The default value for this parameter is set to 8. Yet, 16 is also a reasonable value considering the computational cost of the covariance matrix.

'blk_overlap': A real number showing the amount of overlap between each two neighbouring blocks. Since the edges of a block are invalid due to zero-padding of a block before convolution operation, a minimum amount of overlap is necessary to increase the precision for the covariance matrix of features for an object. The default value for this parameter is set to 0.3. However it can be decreased to even 0.1. There is no need for bigger values, since the only objective for adding this parameter is to avoid the invalid gradient features introduced to the set of features due to invalid edges of the blocks.

4 Experiments and Results

4.1 Setup of Cameras

After a careful investigation on the category of network cameras in the market, we selected the brand D-Link and model type DCS-932L as a suitable and economical solution. Each of these cameras cost around 150$. After being configured with a wireless router once, they only need to be plugged into an electricity cord in order to transfer their images with Wi-Fi connection to the wireless router (preferably n-Router with enough bandwidth for 20 frames per second of 640x480 pixels) and to the software which is provided by D-Link. In this way, we would have much more flexibility for positioning the cameras in an indoor environment compared to webcams.

Figure 4.1: *D-Link DCS-932L, the network camera used for datasets in this thesis*

These cameras can provide us with up to 20 frames per second for 640x480 pixels images with a fairly good quality. Besides, this model supports night vision as well by means of IR LEDs up to around 5 meters of distance from the object. Finally, as the software provided by D-Link is capable of synchronizing the video frames, one can easily create a multi-view set of videos.

In the following paragraphs, the main steps of installation for this camera model will be explained.

Simply connect the electricity cable to the power receptor on the back of the camera and also connect one end of a LAN cable to the ethernet port of the camera and the other end to a wireless router LAN port. Afterwards, run the short installation wizard provided by D-Link on a CD. To view the camera images, one can simply log

in to mydlink.com, choose a device, and start viewing from anywhere by just having access to internet. There is no need to configure the router to open up special ports. All the necessary steps are automatically done by the setup software. Also, it is good to remember the Internet IP addresses for each camera.

Finally, by having the D-Link software installed, we may plan a schedule to start and stop the video recording process by all the cameras at once.

4.2 Datasets in this thesis

In the table 4.1, a brief overview of the datasets used in this thesis is provided. For each dataset, according to challenges faced for the visual tracker, we assign a level of difficulty. Also, each dataset has 3 different views which are disjoint for all except 1st and 2nd scenario. As could be seen, the most difficult case is the 7th scenario in which two objects occlude each other for a short duration of time while continuously changing their pose and appearance. In the later parts, we refer to the number of each scenario, as we discuss about the performance of our visual tracker.

Table 4.1: *List of datasets made by D-Link DSC-932L, their properties*

#Scenario	Difficulty Level	Challenges	#Objects
1	Easy	Affine changes	pair of human eyes
2	Easy	Affine changes	human face
3	Intermediate	Affine, pose change	1 person
4	Intermediate	Affine, pose change	1 person
5	Intermediate	Affine pose change, strong appearance change	1 person
6	Intermediate	Affine pose change, Velocity change	1 person
7	Difficult	Affine changes, partial/short-time full occlusion, pose change, appearance change	2 persons

Figure 4.2: *Different views of the scene in the* ***4th*** *scenario (1st, 31st, 61st frames)*

Figure 4.3: *Movement of the object in the* ***4th*** *scenario, each 30 frames is being displayed (from top-left to bottom-right)*

Figure 4.4: *Different views of the scene in the* ***7th*** *scenario (1st, 31st, 61st frames)*

Figure 4.5: *Movement of the object in the* ***7th*** *scenario, each 30 frames is being displayed (from top-left to bottom-right)*

4.3 Results

In the first part of this section, some tracking results are shown for the 1st and 2nd scenario. Then, in the second part, the basic concepts of epipolar geometry are visualized through the 4th scenario as an introduction to the results for the trajectory. Finally, in the third part, the trajectories of objects movement are shown and explained for the 4th and 7th scenarios. For each part, we provide a brief discussion on the parameter values used and some suggestions to adjust them.

Before going to the first part, some results are shown for the point-based features and the way they are matched and exclusion of outliers from the set of point correspondences by RANSAC or its variants.

SIFT and SURF are quite old and well-known, especially the SIFT, and being efficiently developed in all the popular programming languages in the field of computer vision. There is also a new toolbox in MATLAB 2011b named as Computer Vision within which SURF is fully implemented [51, 52, 53, and 54]. Although the code for these algorithms is ready-to-use, there are few parameters that need to be carefully adjusted for being adapted to each specific scenario. In figure 4.6, just the position of interest points detected and described by SIFT/SURF are shown.

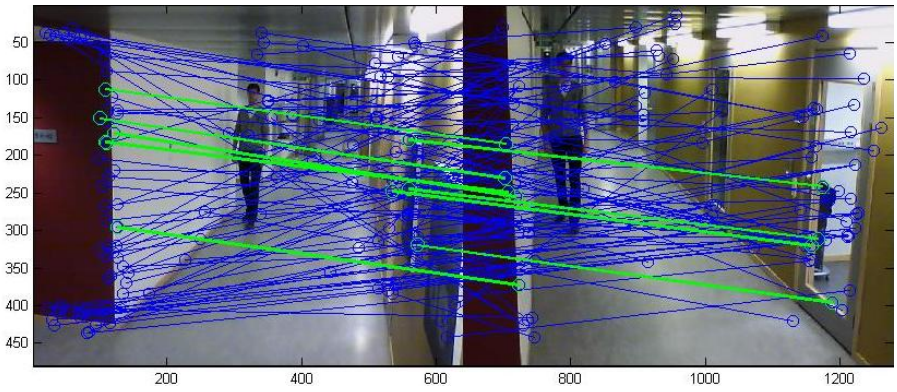


Figure 4.6: *SIFT/SURF point features in the* ***3rd*** *scenario*

As mentioned in the previous paragraph, a few parameters need to be correctly adjusted for SIFT/SURF algorithms for being accurately adopted to each specific scenario. In the following paragraphs, a short discussion is made to clarify the process of adjustment for these set of parameters.

The following parameters need to be adjusted according to the type (rigid/non-rigid) and size of the object (namely, size of the blobs to be detected and matched by SURF) to be tracked and the amount of variations in its appearance during the video (due to occlusion or pose change).

1. ***'hessian_threshold'***: A non-negative scalar which specifies a threshold for selecting the strongest features. Decrease it to return more blobs. The default value for this parameter is 1.0
2. ***'nOctaves'***: An integer scalar, *nOctaves≥1*. Number of octaves to use. Increase this value to detect larger blobs. Recommended values are between 1 and 4. The default value is set to 1 for tracking face and its components such as eyes, nose and mouth.
3. ***'nOctaveLayers'***: Integer scalar, *NumOctaveLayers≥3*. Number of scale levels to compute per octave. Increase this number to detect more blobs at finer scale increments. Recommended values are between 3 and 6. The default value, however, is set to *≥6* to increase the precision of tracking. Each octave spans a number of scales that are analyzed using varying size filters.

Octave	Filter Sizes
1	9x9, 15x15, 21x21, 27x27, ...
2	15x15, 27x27, 39x39, 51x51, ...
3	27x27, 51x51, 75x75, 99x99, ...
4	…

Higher octaves use larger filters and sub-sample the image data. Larger number of octaves will result in finding larger size blobs. *'nOctaves'* should be selected appropriately for the image size. For example, 50x50 image should not require *nOctaves>2*. The number of filters used per octave is controlled by the parameter *'nOctaveLayers'*. To analyze the data in a single octave, at least 3 levels are required.

For the first part, some results are shown for the proposed tracker for the 1st and 2nd scenario. Although the main purpose of this thesis is not the tracking part, we had the chance to (at least try) develop our own visual point-based features tracker. As shown in figure 4.7 and 4.8, one can observe that the tracker is robust enough towards rotation and deformation of the object in a single camera. Besides, the tracker follows the position and orientation of the eyes and face very well.

Figure 4.7: *Tracking results for the 1st scenario*

Figure 4.8: *Tracking results for the 2nd scenario*

There are of course a number of limitations for the current visual tracker. Some of them are listed as below:

1. If the object goes under large pose or appearance change, there are a few numbers of local features to be matched and tracked.
2. Existence of enough local point-based features is directly dependent on the quality of the image and size of the object. In other words, for a small and

poor- resolution object, there might not be enough number of local point-based features to match and track for the visual tracker.

3. The contrast of the object relative to the background affects the number of false matches. This is also known as background clutter and happens when the object appearance is much similar to the area around.
4. Partial/full occlusion can degrade the performance of the tracker in a single view. This issue is described more in details in the last part of this section, where we will give some suggestions to tackle with this problem by using the information from multiple cameras.

Now, for the second part, the basic concepts of epipolar geometry are visualized through the 4th scenario to make the interpretations of the results in the third part easier. As shown with colour lines in figure 4.8, by the fundamental matrix for each pair of views, each point in one view is mapped to a line in the other view. Also, note the intersection of all the colour lines which is a unique point in all the cases. This point is known as the epipole of that view corresponding to a pair of views. The last point is that for e.g. view 1, there are 2 epipoles, one corresponding to the pair of (a):(view1-view2) and the other one corresponding to the pair of (b):(view1-view3).

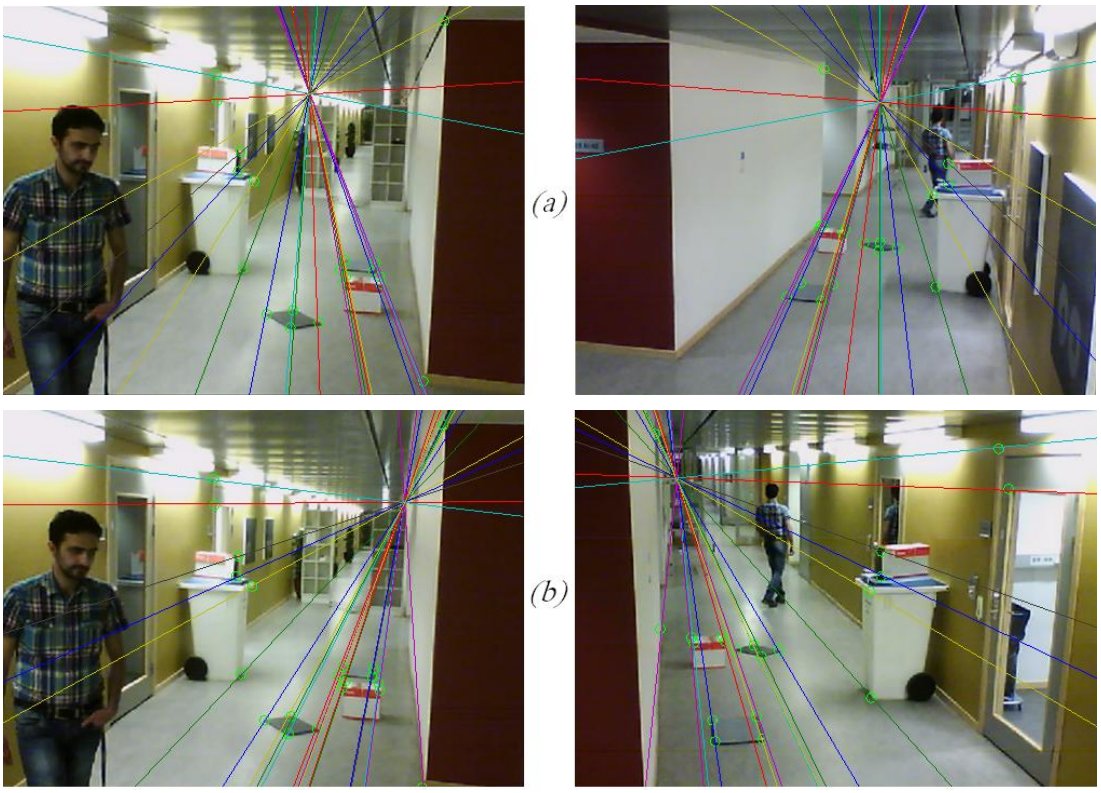


Figure 4.9: *Epipoles and Epipolar lines relating pairs of views (derived by the fundamental matrix)*

So, for the third part we will assume to have a visual tracker giving rise to an accurate bounding box for the object in all the frames and all the views. Our strategy is to consider the centre point of the object bounding box as a representative of the object location in the image plane to project back to the 3D world coordinate system and eventually the ground plane. As a perfect tracker does not exist to our knowledge, there is a possibility of losing the track in one of the views due to occlusion or clutter. In such conditions, we may be able to recover the true position of object in that view by having the object position in the other views and the 3D world coordinate system, as will be explained more in details in the end of this chapter.

From a set of points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}$ with i the number of frames, we can estimate the correspondent position *X* in the 3D world coordinate system, according to section 3.4. In figure 4.10, the first tracked object *(a)* in view 1 of the 7th scenario is shown in green and its position is represented by the location of red star which corresponds to the centre of the green bounding box. The same results are given for view 2 and view 3 of the same scenario in figure 4.11 and 4.12.

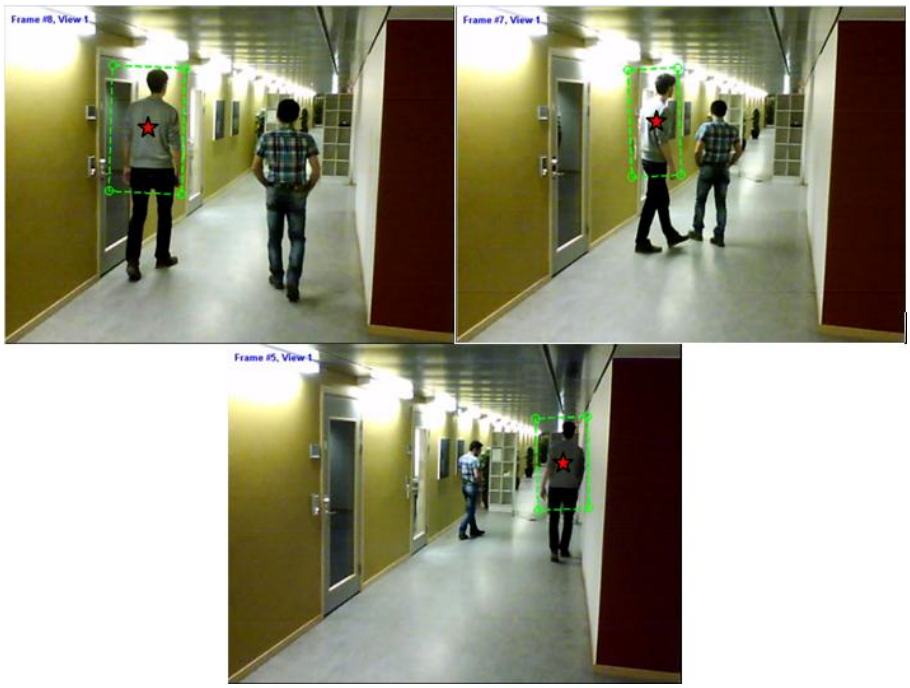


Figure 4.10: *Scenario 7, View 1, Object (a). The object bounding box is shown in green. The red star represents the object position* ($\boldsymbol{x}^{(1)}$)

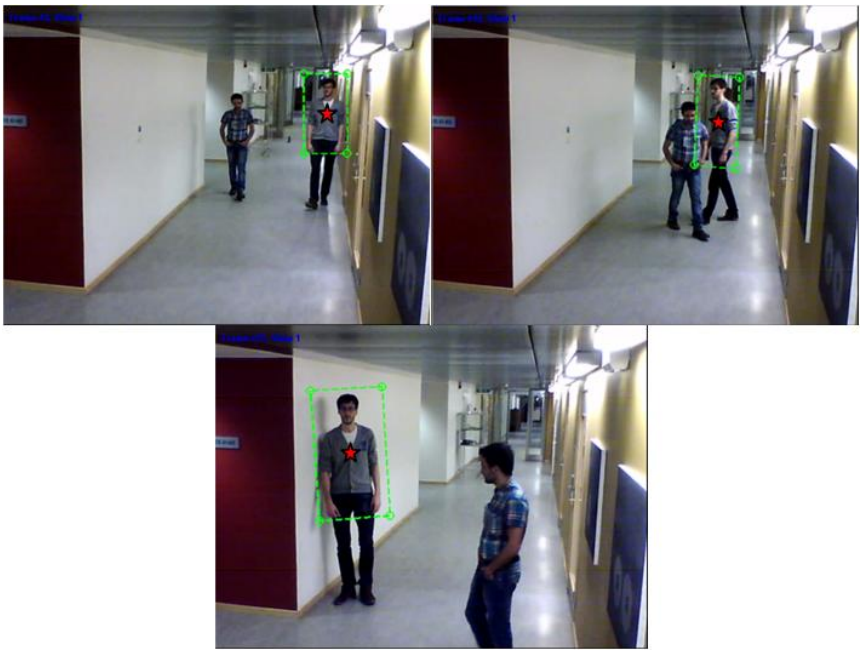


Figure 4.11: *Scenario 7, View 2, Object (a). The object bounding box is shown in green. The red star represents the object position ($\boldsymbol{x}^{(2)}$)*

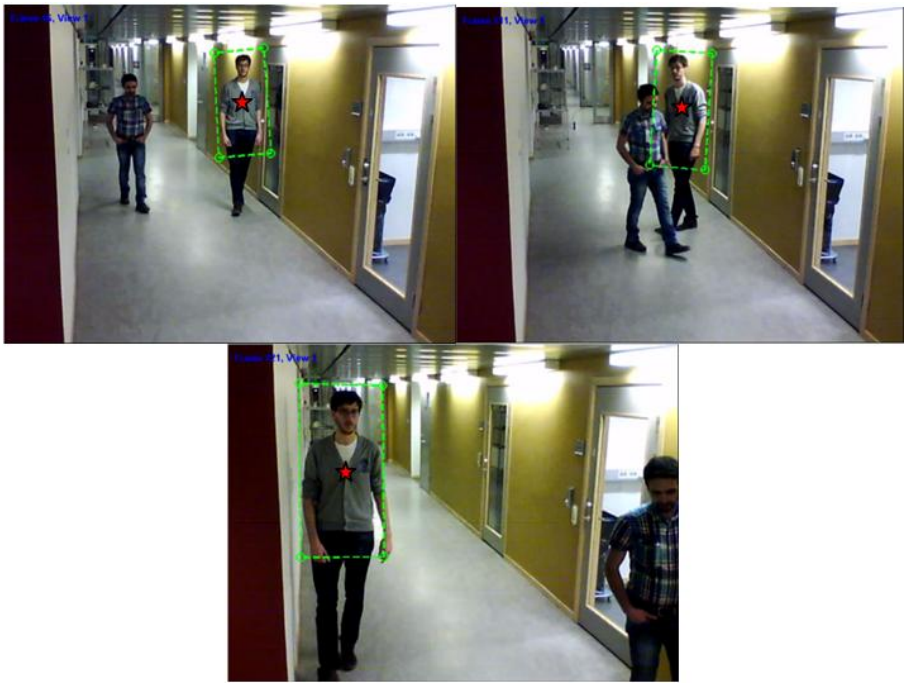


Figure 4.12: *Scenario 7, View 3, Object (a). The object bounding box is shown in green. The red star represents the object position ($\boldsymbol{x}^{(3)}$)*

In figure 4.13, the second tracked object *(b)* in view 1 of the 8th scenario is shown in green and its position is represented by the location of red star which corresponds to the centre of the green bounding box. The same results are given for view 2 and view 3 of the same scenario in figure 4.14 and 4.15.

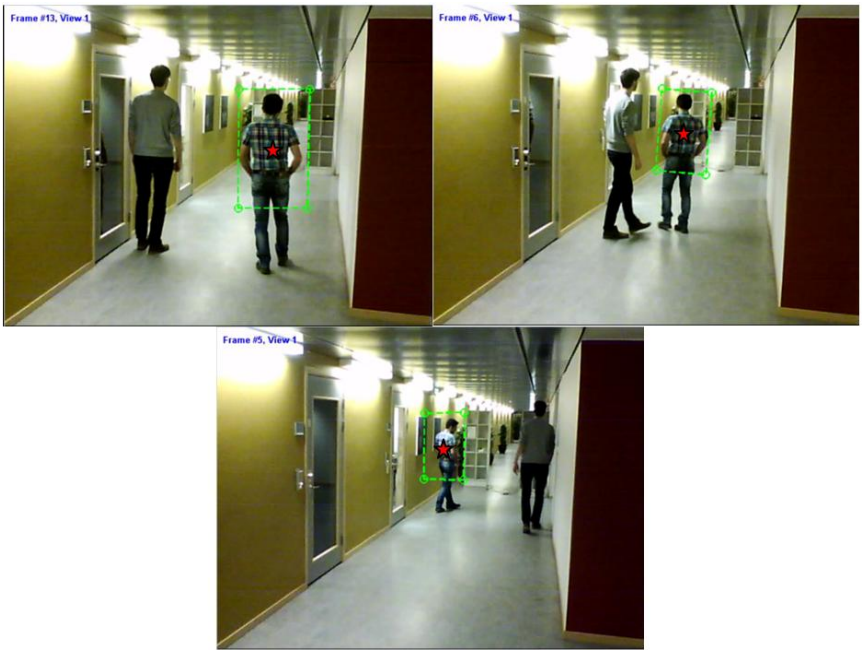


Figure 4.13: *Scenario 8, View 1, Object (b). The object bounding box is shown in green. The red star represents the object position* ($\boldsymbol{x}^{(1)}$)

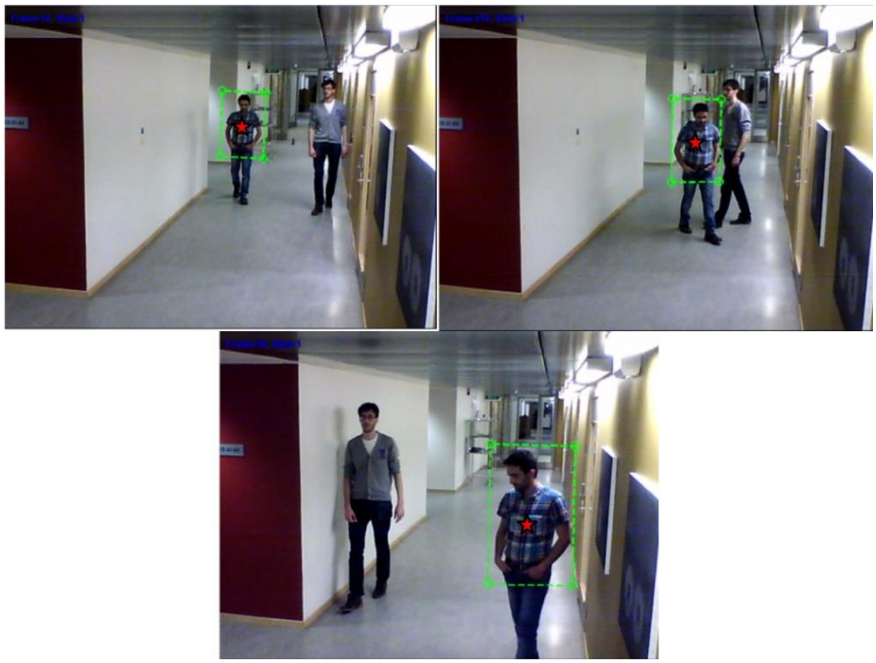


Figure 4.14: *Scenario 8, View 2, Object (b). The object bounding box is shown in green. The red star represents the object position* ($\boldsymbol{x}^{(2)}$)

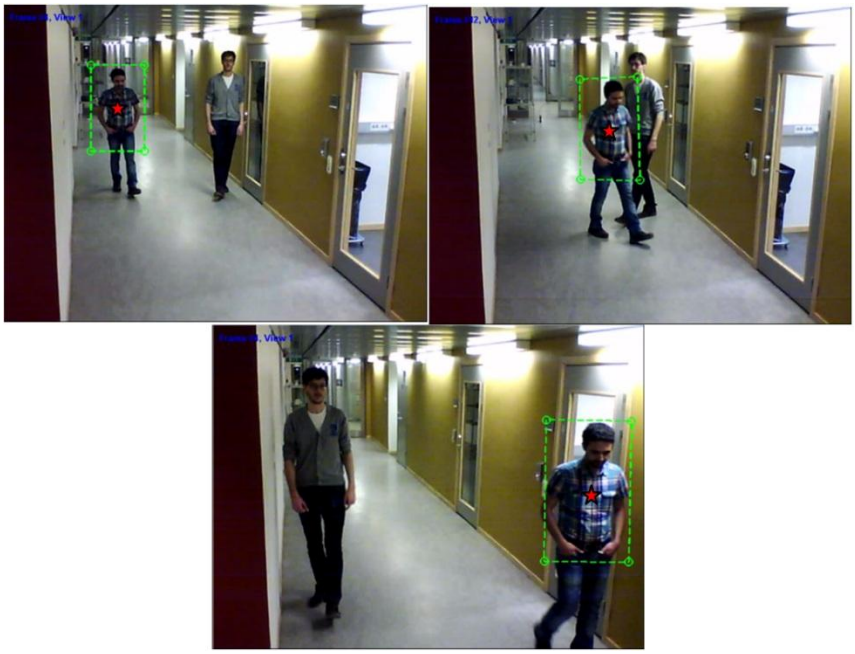


Figure 4.15: *Scenario 8, View 3, Object (b). The object bounding box is shown in green. The red star represents the object position ($\boldsymbol{x}^{(3)}$)*

The fundamental matrix is calculated using 20 static points manually selected from all the views. This set of points is the input to the norm8point algorithm described in section 2.2.4. This method gives rise to better results for the trajectory.

In the following figures, the trajectories of the object on the ground plane in scenario 4 are shown. In this scenario, there is a single object to be tracked which corresponds to a human body. The pedestrian sometimes walks from the right side to the left side of the corridor and sometimes doing a circular motion. The movements of the pedestrian confirm the trajectory to some extent. Due to different sources for error, these variations from real movements can be described.

- ✓ **Note 1:** As sometimes, the object position overlaps for some frames, it is not visually appealing to show the trajectory over the whole frame range of the video. Hence, the trajectory is only visualized within a short frame range of the video.

- ✓ **Note 2:** For more careful consideration of object movement, the changes of X and Y versus frame number is also shown.

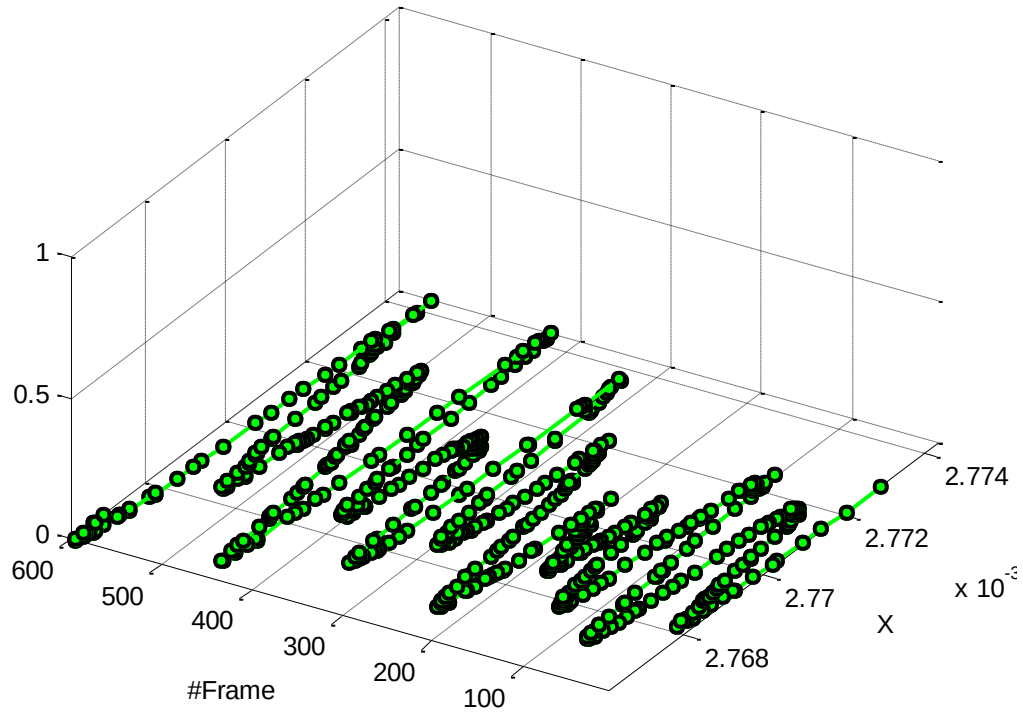


Figure 4.16: *Scenario 4, The trajectories of object movement on the ground plane are shown. X of the object on the ground plane is plotted versus the frame number (Time).*

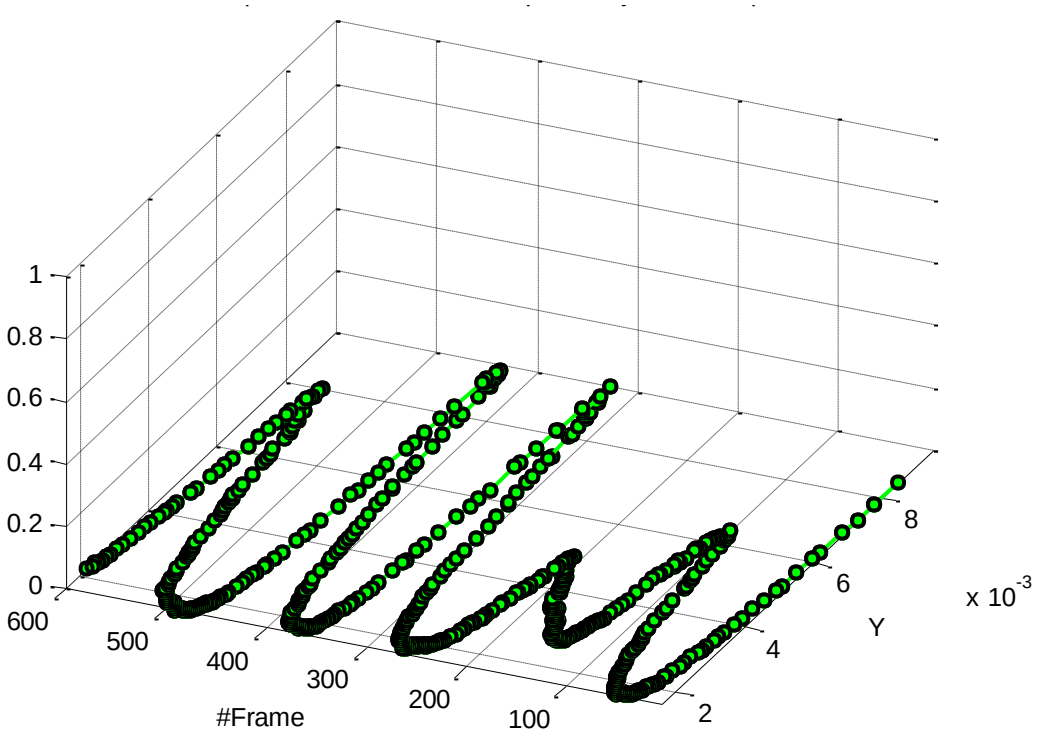


Figure 4.17: *Scenario 4, The trajectories of object movement on the ground plane are shown. Y of the object on the ground plane is plotted versus the frame number (Time).*

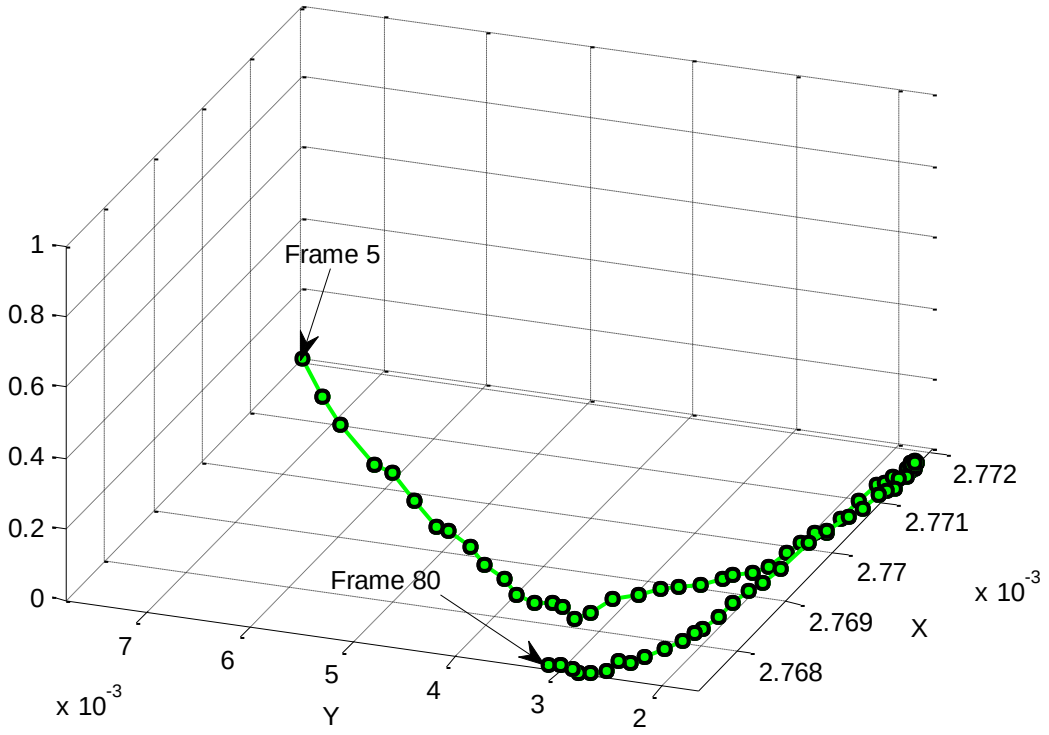


Figure 4.18: *Scenario 4, The trajectories of object movement on the ground plane are shown from frames 5 to 80.*

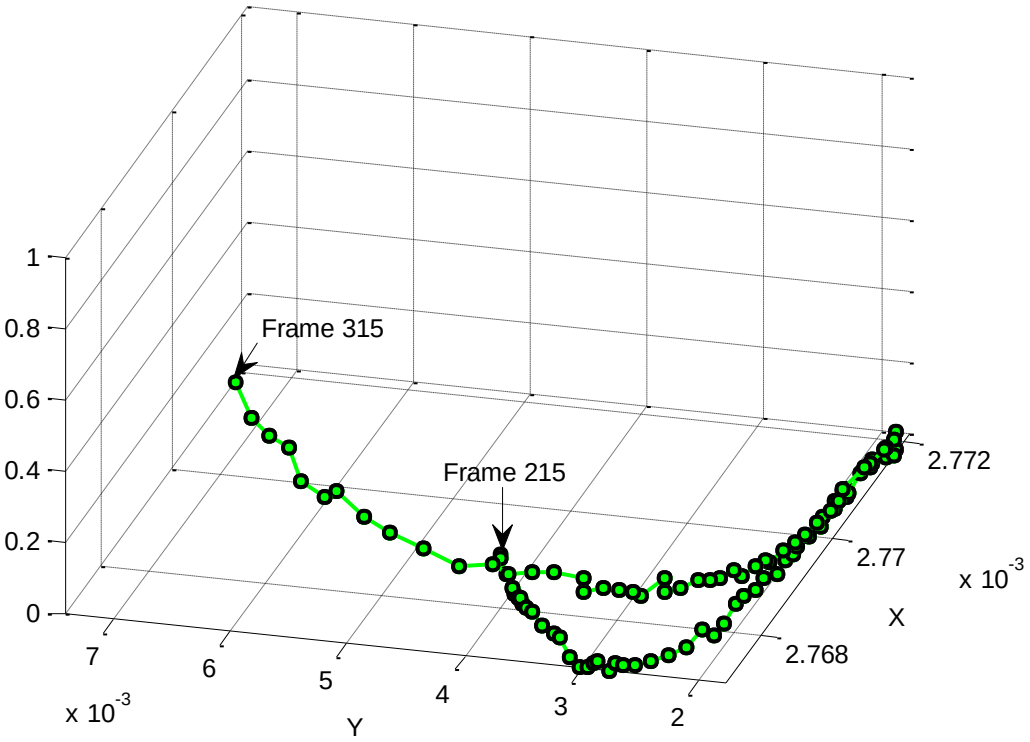


Figure 4.19: *Scenario 4, The trajectories of object movement on the ground plane are shown from frames 315 to 215.*

The same as above, in the following figures, the trajectories of 2 moving objects on the ground plane in scenario 7 are shown. The point of intersection for the two trajectories represents the moment in which two objects cross (occlude) each other. In particular in this scenario, the two objects start their movements from different positions (left, right) and they are within the closest distance to each other in the frame number 198 (the moment in which they cross each other). As we can see from the figure 4.24, the objects have more distance from each other before and after that frame number. This fact is confirmed once more in figure 4.27. In short, the trajectory confirms the real movements.

The reason why these trajectories are not so visually appealing (not consistent with the real shape of object movement on the real ground plane) is because of the projective ambiguity in the reconstruction of $\mathbf{X}$ from $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, $\mathbf{x}^{(3)}$.

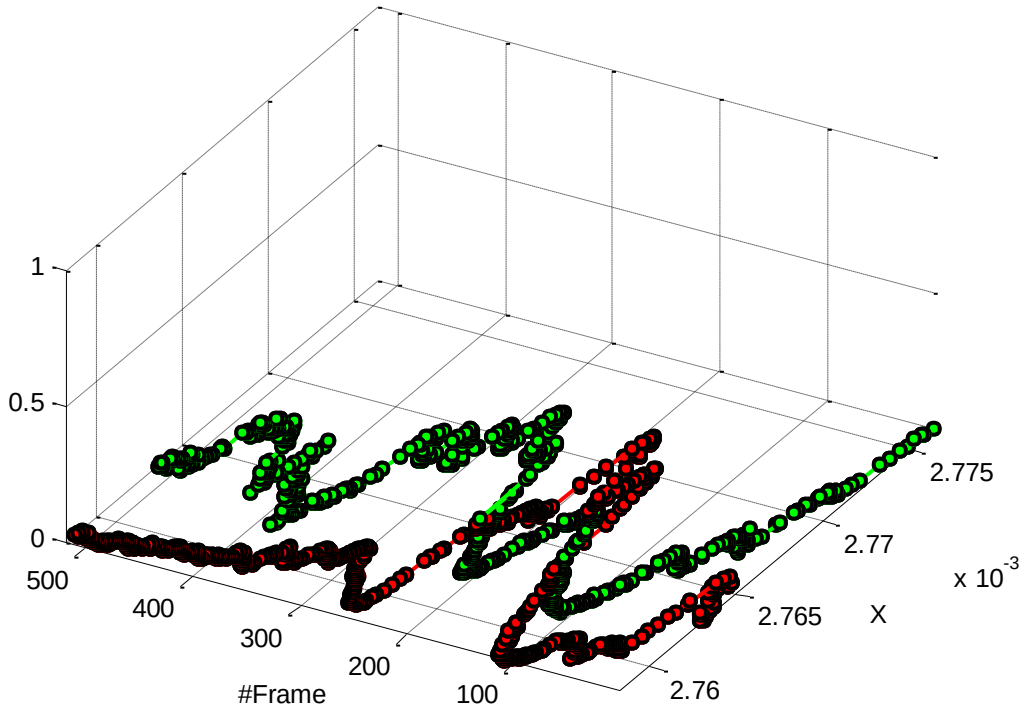


Figure 4.20: *Scenario 7, The trajectories of objects movement on the ground plane are shown: (a):red, (b):green. X of the object on the ground plane is plotted versus the frame number (Time).*

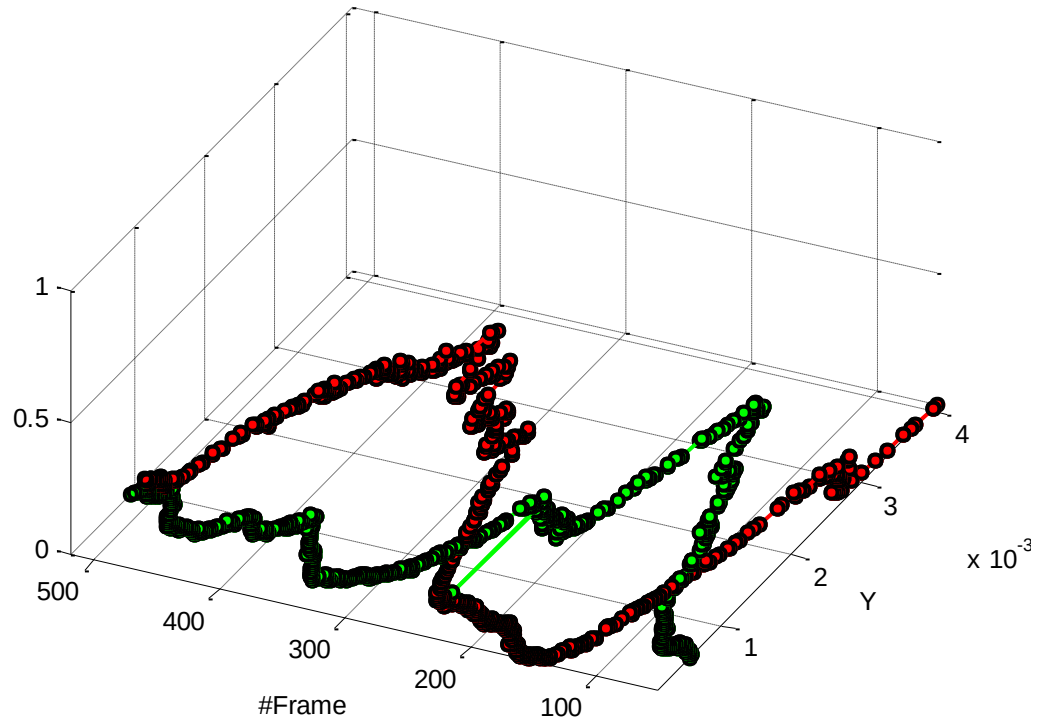


Figure 4.21: *Scenario 7, The trajectories of objects movement on the ground plane are shown: (a):red, (b):green. Y of the object on the ground plane is plotted versus the frame number (Time).*

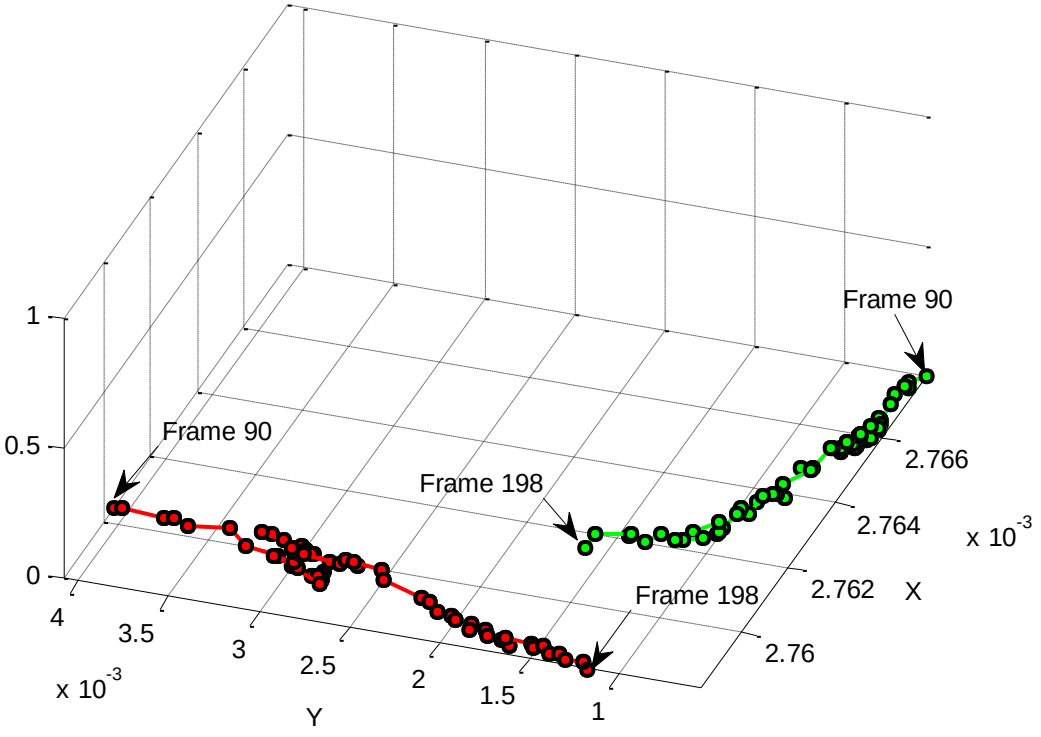


Figure 4.22: *Scenario 7, The trajectories of objects movement on the ground plane are shown: (a):red, (b):green both from frame 90 to frame 198.*

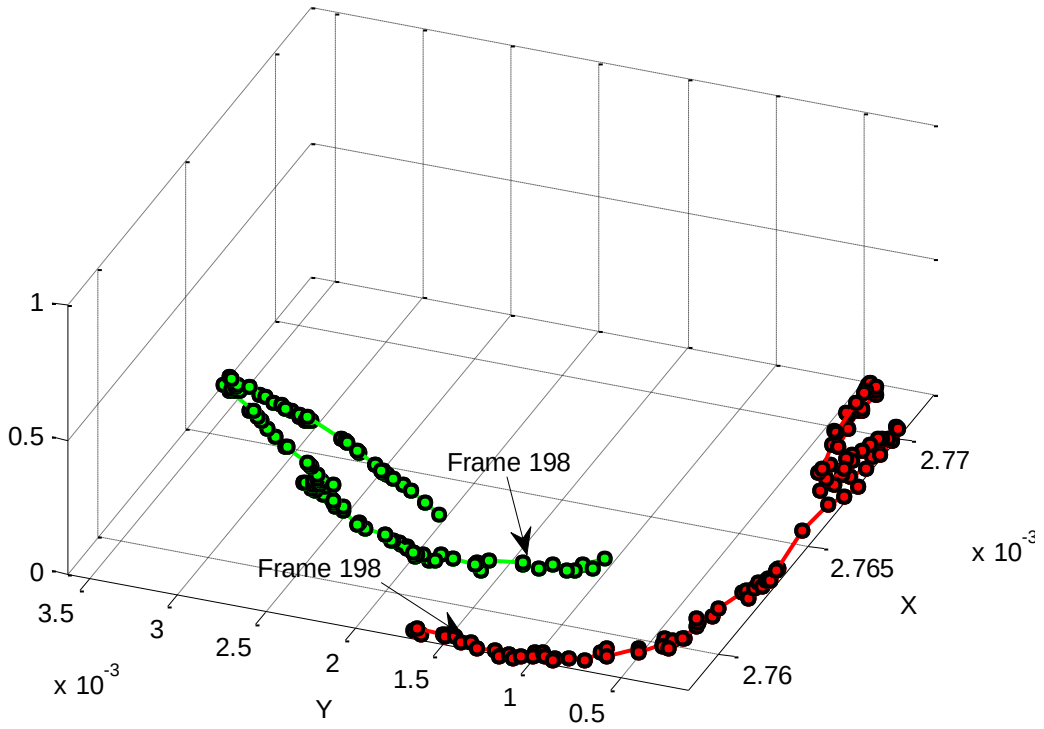


Figure 4.23: *Scenario 7, The trajectories of objects movement on the ground plane are shown: (a):red, (b). In the frame 198 there is the moment where the two trajectory are closest each other.*

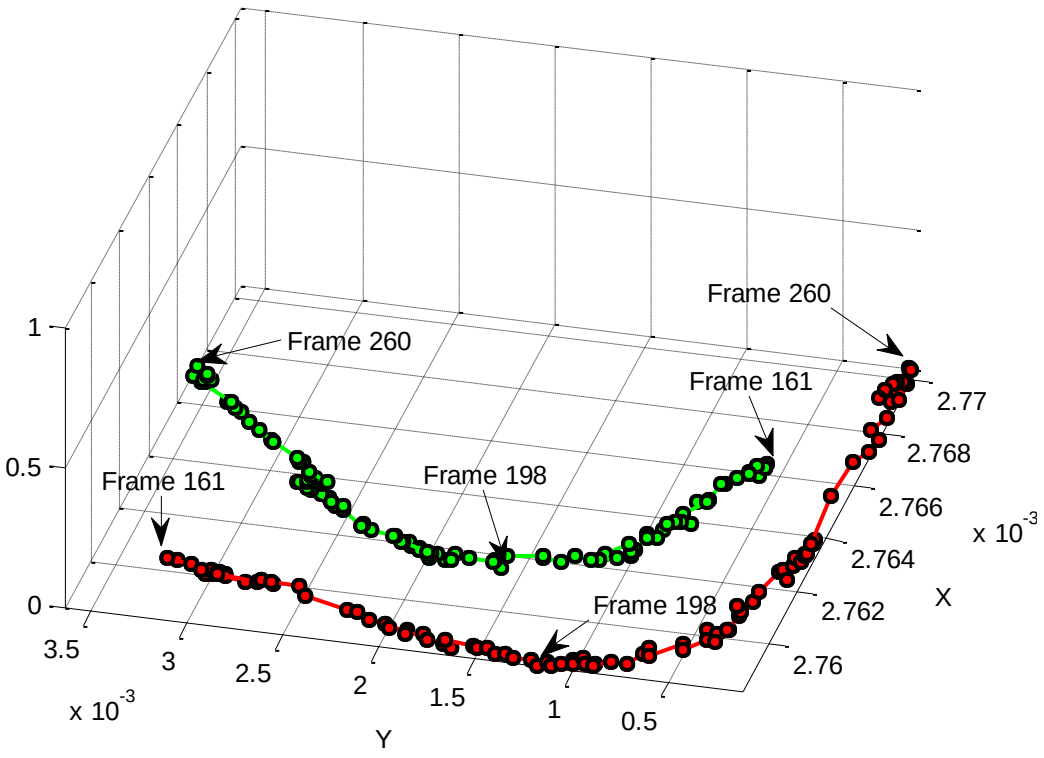


Figure 4.24: *Scenario 7, The trajectories of objects movement on the ground plane are shown: (a):red, (b):green both from frame 161 to frame 260.*

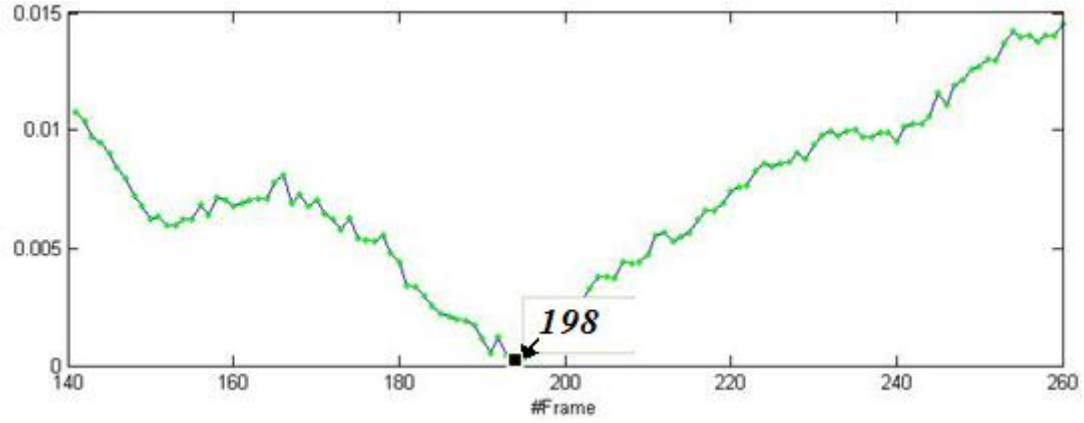


***Figure 4.25**: Scenario 7, Euclidean distance between the points in the two trajectories, minimum distance happens at frame 198 (the moment in which the 2 objects cross each other in the scene).*

5 Conclusion and Future Work

"The important thing is not to stop questioning. Curiosity has its own reason for existing."
(Albert Einstein)

This thesis work proposes a method for estimation of the trajectory for object movements on the ground plane from point correspondences between disjoint uncalibrated cameras. In short, the idea is to help the tracker in one view by using the information from other views, since all the decent single-view trackers in the field would get lost in case of full and long-duration occlusion.

There were several challenges faced in this work, some expected while some others unexpected. The most important challenge was to correctly make a set of point correspondences between multiple disjoint views in which the objects have completely different appearance and position. There needs to be more advanced and intelligent methods for relating different parts of the scene among different views. By those relationships, a computer system may better understand the scene. Then, by relying on this knowledge, one may get better use of interest point detectors. Eventually, there must be a better criterion for matching interest points in 2 or more views instead of just relying upon a histogram of gradients-based feature vector in a neighbourhood around the interest point.

Several challenges are about tracking the object within the current point-based feature tracking with multiple models. All of these challenges act as a stimulus to go on and to try more ideas. In summary, based on our knowledge and experience, the current tracking scheme could not be applied to long and complicated scenarios. Specially, even in the case of partial occlusion or appearance change, the tracker would easily fail.

The current methodology for deriving the trajectory leads to some shapes which is not essentially consistent with the real shape of movements in the real world (i.e. not visually appealing to the user). Yet, there are two possible advantages for having such trajectory of object movements on the ground plane:

1. One may be able to recover the true position of the object in the occluded view by having the 3D position of the object and its position in other views. Hence, the visual tracker will be able to continue the track of object in that view, although it remains occluded for a while.

2. One may combine 1 with the occlusion metric to recognize when the object becomes occluded or visible again. In this way, the tracker may understand when and where the object becomes visible or occluded again.

For the 1st advantage, one should think of the inverse of reconstruction. In other words, given X and $\hat{x}_i, \hat{x}_i'$ the goal is to get $\hat{x}_i''$ which is an estimation for the true position of the object in 3rd view in which the object is partially/fully occluded and the tracking is probably lost or not accurate enough. Besides, if the 2nd advantage is integrated, the tracker would be able to recognize if the occlusion has happened or not and our vision system would be more intelligent in such case.

For the 2nd advantage, one idea could be computing a 3D appearance model for the object and scene by 3D reconstruction. Such appearance models should be able to precisely describe the object appearance seen from different views and in different lighting conditions. In such case, the object pose or appearance change would not be recognized as occlusion and hence the algorithm may easily deal with these changes by having such 3D model.

Another idea should be building a synthetic top-view image, i.e. an image that the real scene is reconstructed as viewed from top of the ground with all the cameras within the field of view. With this synthetic image, one may compute the 2D planar homography between camera image planes and this synthetic top-view image to be able to project back the position of the object onto the so-called ground plane in a much better and more visually appealing way. However, this solution may limit the applicability of the algorithm to many real-life scenarios.

Besides all above, an alternative way for making relationship between multi-view images of a scene is the trifocal tensor. As noticed in [1], the trifocal tensor could be considered as an extension of the fundamental matrix for relating 3 views together. By having the trifocal tensor, one may be able to recover fundamental, essential, camera matrices and a lot more. Also, by using the trifocal tensor, the point correspondences in a pair of views could be transferred and related to the correspondent point in the third view. Most interestingly, the trifocal tensor can be computed from only image point correspondences and without having any information about the cameras. In [1] some more explanatory ideas are offered for the interested reader.

6 References

- **Books:**

[1] Richard Hartley, Andrew Zisserman, “Multiple View Geometry in Computer Vision”, 2nd Ed., Cambridge University Press, March 2004.

[2] Peter Corke, "Robotics, Vision and Control: Fundamental Algorithms in MATLAB", 1st Edition, Springer Tracts in Advanced Robotics, 2011.

[3] David Forsyth, Jean Ponce, “Computer Vision - A Modern Approach”, Prentice Hall, 2003.

[4] Shapiro, Linda and George C. Stockman, “Computer Vision”, 2001.

[5] Trucco, Alessandro Verri, “Introductory Techniques for 3-D Computer Vision”, 1998.

[6] Richard Szeliski, “Computer Vision: Algorithms and Applications”, Springer, 2010.

[7] Mobarak Shah, “Fundamentals of Computer Vision”, Computer Science Department, University of Central Florida, 1997.

[8] Christopher M. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2006.

[9] Abraham Berman, Naomi Shaked-Monderer, “Completely Positive Matrices”, World Scientific Publishing Co. Pte. Ltd., 2003.

- **Papers:**

[10] D. Lowe, "Distinctive image features from scale-invariant keypoints", International Journal in Computer Vision, Vol. 60, Issue 2, p91-110, 2004.

[11] C. Harris, M.J. Stephens, "A combined corner and edge detector", In Proceedings of 4th Alvey Vision Conference, p189–192, 1988.

[12] K. Mikolajczyk, C. Schmid, "Scale and affine invariant interest point detectors", International Journal of Computer Vision, Vol. 60, Issue 1, p63-86, 2004.

[13] K. Mikolajczyk, C. Schmid, "A performance evaluation of local descriptors", IEEE Transactions on Pattern Analysis & Machine Intelligence, Vol. 27, Issue 10, p1615–1630, 2005.

[14] C. Schmid, R. Mohr, and C. Bauckhage, “Evaluation of interest point detectors”, International Journal of Computer Vision, Vol. 37, Issue 2, p151–172, June 2000.

[15] Bay, H., Tuytelaars, T., and Gool, L. V., “Surf: Speeded up robust features”, In Proceedings of the 9th European Conference on Computer Vision, Vol. 3951, Part 1, Springer, p404-417, 2006.

[16] M.A. Fischler and R.C Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartographys”, Communications of the ACM, vol. 24, pp. 381-395, 1981.

[17] Petter Strandmark, and Irene Y.H. Gu, “Joint Random Sample Consensus and Multiple Motion Models for Robust Video Tracking”, In Proceedings of the 16th Scandinavian Conference on Image Analysis, 2009.

[18] Sebastian Haner, Irene Y.H. Gu, "Combining Foreground / Background Feature Points and Anisotropic Mean Shift For Enhanced Visual Object Tracking", 20th International Conference on Pattern Recognition (ICPR), pp.3488-3491, 2010.

[19] N. Dalal, B. Triggs, “Histograms of Oriented Gradients for Human Detection”, CVPR, 2005.

[20] N. Dalal, B. Triggs, C. Schmid, “Human Detection Using Oriented Histograms of Flow and Appearance”, European Conference on Computer Vision, 2006.

[21] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, B. Girod,, “CHoG-Compressed Histogram of Gradients-A Low Bit-Rate Feature Descriptor”, IEEE Conference on Computer Vision and Pattern Recognition, p2504-2511, 20-25 June 2009.

[22] Xi Li, W. Hu, Z. Zhang, X. Zhang, M. Zhu, J. Cheng, "Visual tracking via incremental Log-Euclidean Riemannian subspace learning", IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2008, pp.1-8, 23-28 June 2008.

[23] F. Porikli, O. Tuzel, P. Meer, "Covariance Tracking using Model Update Based on Lie Algebra", Computer Vision and Pattern Recognition, IEEE Computer Society Conference, vol.1, pp. 728- 735, 17-22 June 2006.

[24] V. Arsigny, P. Fillard, X. Pennec, N. Ayache, “Geometric Means in a Novel Vector Space Structure on Symmetric Positive-Definite Matrices”, SIAM Journal on Matrix Analysis and Applications, 29(1):328-347, 2006.

[25] D. Ross, J. Lim, and M. Yang, “Adaptive Probabilistic Visual Tracking with Incremental Subspace Update”, Eighth European Conference on Computer Vision, 2004.

[26] David A. Ross, J. Lim, R. Lin, M. Yang, “Incremental Learning for Robust Visual Tracking”, International Journal of Computer Vision, Volume 77 Issue 1-3, May 2008.

[27] David Ross, J. Lim, R. Lin, M. Yang, “Incremental Learning for Visual Tracking”, In Advances in Neural Information Processing Systems, p793-800, MIT Press, 2004.

[28] J. Yang, D. Zhang, A. F. Frangi, J. Yang, “2DPCA-A New Approach to Face Representation and Recognition”, IEEE Transactions on Pattern Analysis & Machine Intelligence, 2004.

[29] D. Zhang, Z. Zhou, “(2D)2PCA: 2-Directional 2-Dimensional PCA for Efficient Face Representation and Recognition”, 2005.

[30] T. Wang, Irene Y.H. Gu, P. Shi, “Object Tracking using Incremental 2D-PCA Learning and ML Estimation”, IEEE International Conference on Acoustics, Speech and Signal Processing, 2007.

[31] M. Isard, A. Blake, “CONDENSATION – conditional density propagation for visual tracking”, International Journal of Computer Vision, 1998.

[32] Ingrid Carlbom, Joseph Paciorek, "Planar Geometric Projections and Viewing Transformations", ACM Computing Surveys (CSUR), Vol. 10, Issue 4, December 1978.

[33] Z. Zhang , T. Kanade, “Determining the Epipolar Geometry and its Uncertainty: A Review”, International Journal of Computer Vision, 1998.

[34] X. Armangué, J. Salvi, “Overall View Regarding Fundamental Matrix Estimation”, Image and Vision Computing, Volume 21, Issue 2, P205-220, 2003.

[35] A. C. Sankaranarayanan, R. Chellappa, “Optimal Multi-View Fusion of Object Locations”, Proceedings of the IEEE, WMVC, January 2008.

[36] A. C. Sankaranarayanan, A. Veeraraghavan, R. Chellappa, “Object Detection, Tracking and Recognition for Multiple Smart Cameras”, Proceedings of the IEEE, Vol. 96, Issue 10, October 2008.

[37] F. Fleuret, J. Berclaz, R. Lengagne, P. Fua, “Multi-Camera People Tracking with a Probabilistic Occupancy Map”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 30, Issue 2, p267-282, February 2008.

[38] Saad M. Khan, M. Shah, "A Multiview Approach to Tracking People in Crowded Scenes using a Planar Homography Constraint", In Proceedings of ECCV, 2006.

[39] S.M. Khan, M. Shah, "Tracking Multiple Occluding People by Localizing on Multiple Scene Planes", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 31, Issue 3, p505-519, March 2009.

[40] O. Javed, Z. Rasheed, K. Shafique, M. Shah, "Tracking across Multiple Cameras with Disjoint Views", Ninth IEEE International Conference on Computer Vision, Vol. 2, p952-957, 2003.

- **Scientific Reports (Master Thesis, PhD Dissertation, Technical Reports):**

[41] Kevin Cannons, "A Review of Visual Tracking", Technical Report, York University, September 16, 2008.

[42] Alper Yilmaz, Omar Javed, Mubarak Shah, "Object tracking: A survey", ACM Computing Surveys (CSUR), Vol. 38, Issue 4, 2006.

[43] Konstantinos G. Derpanis, "The Harris Corner Detector", October 27, 2004.

[44] Petter Strandmark, "Feature Tracking With Multiple Models", Master's Thesis, Chalmers University of Technology, October 27th 2008.

[45] Hao Wu, Aswin C. Sankaranarayanan, Rama Chellappa, "Online Empirical Evaluation of Tracking Algorithms", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, Issue 8, August 2010.

[46] Aswin C. Sankaranarayanan, "Robust and Efficient Inference of Scene and Object Motion in Multi-Camera Systems", PhD Dissertation, 2009.

[47] Navneet Dalal, "Finding People in Images and Videos", PhD Dissertation, 2006.

[48] Jason Denton, "2 Dimensional Projective Point Matching", PhD Dissertation, July 2002.

[49] H. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover", Technical Report, Carnegie-Mellon University, Robotics Institute, 1980.

[50] D. M. Gavrila, "The visual analysis of human movement: A Survey", Computer Vision and Image Understanding, 73(1):82–98, 1999.

- **Other Sources (Web Sites, MATLAB Toolboxes, Source Codes):**

[51] SIFT Keypoint Detector, David Lowe, http://www.cs.ubc.ca/~lowe/keypoints/, Last Viewed 2011-02-10.

[52] SIFT For MATLAB, A. Vedaldi, http://www.vlfeat.org/~vedaldi/code/sift.html, Last Viewed 2011-02-10.

[53] SURF For MATLAB, Peter Strandmark, http://www.maths.lth.se/matematiklth/personal/petter/surfmex.php.

[54] The OpenSURF Computer Vision Library, Chris Evans, http://www.chrisevansdev.com/computer-vision-opensurf.html, Last Viewed 2011-02-09.

[55] MATLAB Functions for Multiple View Geometry, http://www.robots.ox.ac.uk/~vgg/hzbook/code/, Last Viewed 2011-02-16.

[56] Functions Supporting Projective Geometry, Peter Kovesi: http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/index.html

[57] Iterative methods for computation of Fundamental matrix, Omid Aghazadeh: http://www.mathworks.com/matlabcentral/fileexchange/27541-fundamental-matrix-computation

[58] Homography: http://mmlab.disi.unitn.it/wiki/index.php/2D_Homography.

[59] MATLAB Teaching Codes, http://web.mit.edu/18.06/www/Course-Info/Tcodes.html

[60] PETS 2007 Dataset, Calibration, Data, C++ Code, http://www.cvg.rdg.ac.uk/PETS2007/data.html

Appendix A: Homogeneous vs. Inhomogeneous Coordinate System

A point in *n*-dimensional Euclidean space$x \in \mathbb{R}^n$ is represented by a coordinate vector $(x_1, x_2 \cdots x_n)$. The corresponding point in homogeneous coordinates, or the projective space $\tilde{x} \in \mathbb{P}^n$ is represented by a coordinate vector $(\tilde{x}_1, \tilde{x}_2 \cdots \tilde{x}_{n+1})$. The Euclidean coordinates are related to the projective coordinates by

$$x_i = \frac{\tilde{x}_i}{\tilde{x}_{n+1}}, i = 1 \cdots n$$

Conversely a homogeneous coordinate vector can be constructed from a Euclidean coordinate vector by

$$\tilde{x} = (x_1, x_2 \cdots x_n, 1)$$

and the tilde is used to indicate that the quantity is homogeneous.

The extra degree *of freedom* offered by projective coordinates has several advantages. It allows points and lines at infinity, known as ideal points and lines, to be represented using only real numbers. It also means that scale is unimportant, that is $\tilde{x}$ and $\widetilde{x'} = \alpha\tilde{x}$ both represent the same Euclidean point for all $\alpha \neq 0$. We express this as $\tilde{x} \cong \widetilde{x'}$.

Points in homogeneous form can also be rotated with respect to a coordinate frame and translated simply by multiplying the homogeneous coordinate by an $(n+1) \times (n+1)$ homogeneous transformation matrix.

Homogeneous vectors are important in computer vision when we consider points and lines that exist in a plane – a camera's image plane. We can also consider that the homogeneous form represents a ray in Euclidean space, and the relationship between points and rays is at the core of the projective transformation.

In $\mathbb{P}^2$ a line is defined by a 3-tuple, $\tilde{l} = (l_1, l_2, l_3)^T$, not all zero, and the equation of the line is the set of all points

$$\tilde{l}^T\tilde{x} = 0$$

which expands to $l_1x + l_2x + l_3x = 0$ and can be manipulated into the more familiar representation of a line. Note that this form can represent a vertical line,

parallel to the *y*-axis, which the familiar form $y = mx + c$ cannot. This is the point equation of a line. The non-homogeneous vector (l_1, l_2) is a normal to the line, and $(-l_2, l_1)$ is parallel to the line.

A duality exists between points and lines. A point is defined by the intersection of two lines. If we write the point equations for two lines ${\tilde{l}_1}^T \tilde{p} = 0$ and ${\tilde{l}_2}^T \tilde{p} = 0$ their intersection is the point

$$\tilde{p} = \tilde{l}_1 \times \tilde{l}_2$$

and is known as the line equation of a point. Similarly, a line joining two points $\tilde{p}_1$and $\tilde{p}_2$ is given by the cross-product

$$\tilde{l}_{12} = \tilde{p}_1 \times \tilde{p}_2$$

This is an ideal point since the third coordinate is zero – the equivalent Euclidean point would be at infinity. Projective coordinates allow points and lines at infinity to be simply represented and manipulated without special logic to handle the special case of infinity.

The distance from a point $\tilde{p}$ to a line $\tilde{l}$ is

$$d = \frac{\tilde{l}^T \tilde{p}}{p_3 \sqrt{l_1^2 + l_2^2}}$$

In the projective space $\mathbb{P}^3$ a duality exists between points and planes: three points define a plane, and the intersection of three planes defines a point [2].