Abstract

This project presents an application for image matching between different views and scales of an object within an image. A new robust method is proposed collaborating SIFT [1], Cross Correlation Matching [25] and RANSAC [9] algorithms, to obtain a problem solution. These algorithms are replicated and optimised in terms of performance and efficiency, and critically analysed in the following chapters. The proposed method allows concurrent decision if one or several objects are common to the two images, extracting distinctive invariant features from the images and estimating the corresponding matching rate. The features are invariant to image scaling and rotation and may be affected by distortion, noise and different illumination. The features are highly distinctive and can be correctly matched with high probability against a large database of features.

This project also describes the use of the proposed method as a way of simplifying the RS company component ordering procedure. The RS Electronics Company sells a variety of electronic components, where component orders are placed using the parts' serial numbers. The major setback is that most of the times the component serial number cannot be retrieved.

By using the project application, the client will be able to place an order by sending a photo of the component. The client is required to take some photos of the component and upload them in the company online system that uses the proposed application. The company system then processes the image, attempts to match it with one or more images from the components' database. At the end the application provides to the client a detailed feedback report informing whether the component was found.

Experimental results on real images verify the effectiveness of the proposed method in respect of accuracy and speed, on images with various characteristics.

Table of Contents

List of Symbols

$D(x, y, \sigma)$	Scale-space function.
$d^2_{I,k}(i, j)$	Euclidean distance measure.
$D(\hat{x})$	Extremum function.
e	Probability that is an outlier.
F	Fundamental matrix.
f1, f2	SIFT feature arrays for image 1 and 2.
$G(x, y,\sigma)$	Gaussian scale function.
g_σ	Gaussian kernel isotropic.
H	Hessian matrix.
$I(x)$	Input image.
$I(x, y)$	Input image.
k	Constant multiplicative factor.
$L(x,y)$	Image sample.
$L(x, y,\sigma)$	Image scale space function.
m	Magnification factor.
$m(x,y)$	Gradient magnitude.
N1 , N2	Normalised matrices for image 1 and 2.
O	Number of octaves.
$O(x, y)$	Output of the cross correlation function.
o_{min}	First octave index.
p	Probability.

R	Ratio between the two eigenvalues.
r_c	Correlation coefficient.
s	Sample of data points.
S	Number of sub-levels of each octave.
S_i	Image frames (keypoints) data set.
t	Threshold.
$\hat{x}$	Location of the extremum.
α	Eigenvalue with the largest magnitude.
β	Eigenvalue with the smaller magnitude.
$\theta\ (x,y)$	Orientation theta.
N'	Number of RANSAC trials.
σ	Sigma scale.
σ_0	The base smoothing.
$\sigma_{n,}$	The nominal smoothing level for the input image.
σ_w	Gaussian window deviation.
$\sigma^2\nabla^2 G$	Scale- normalized Laplacian of Gaussian.

Nomenclature

CS	Consensus Set
DFD	Data Flowchart Diagram
DoG	Difference of Gaussian
FIR	Finite Impulse Response
GSS	Gaussian Scale Space
MbC	Match by Correlation
MMS	Minimal Sample Sets
MR	Matching Rate
NaN	Not a Number
NCC	Normalised Cross Correlation
NNA	Nearest Neighbour Algorithm
PM	Putative Matches
RANSAC	Random Sample Consensus
SIFT	Scale Invariant Feature Transform

1. Introduction

The digital image processing field is in great demand in the industry and the research area. During the end of the last century, mankind started to make its first steps into the digital image world [33]. Within a short period of time, humans abandoned film based cameras and started using digital ones, which are more compact, easy to use and which offer better resolution compared to the price. Ever since technology switched to digital cameras, an investment boost has occurred in the field of research and development of digital image processing. Image processing was employed by photographers and related jobs some time before the advent of the digital camera, as a tool to edit photos and images. With the use of a digital camera, it became easier to get an image without the interaction of professional and digital image processing software. Hence, it could be used by everyone, without requiring any professional knowledge. Very soon, digital image processing became a very popular subject and today is being taught in schools and universities. Image processing can moreover be used at an advanced level; to extract image information and details or in the context of different applications which could not be implemented with a different approach.

A representative example is an autonomous surveillance system, which was previously based on sensors to scan the environment but which now uses cameras and image processing as the main tool for recognition. Thus, advanced algorithms are designed to serve image processing applications. Image processing algorithms are now embedded in hardware to improve the performance of the digital image process, such as that of a Smartphone or a digital camera which utilise a face recognition algorithm to identify and focus on a human face at the moment of trying to capture an image. In a surveillance system, an edge detection algorithm may be used to recognise whether a human has crossed into the protected area. Such application may appear to be a simple one which performs an easy and simple task. Nevertheless, advanced algorithms using high level mathematical calculations are being implemented and run behind this application. Furthermore, a combination of these algorithms is widely applied to enhance the performance and the broad application abilities.

Object matching in digital image processing is a very complex problem with a variety of uses despite the fact that it may look simple at first sight. The idea behind the object matching concept is to create a system algorithm to identify whether two images have similarities as well as the degree of such similarity. Imagine two different images of the same object, where each image is captured from a different distance and different perspective. The result theoretically should be positive, since the same object exists in both images. There are several obstacles that may affect the result, such as image noise and distortion which will affect images individually. Another issue in object matching problem solutions is the algorithm performance related to speed, correctness and accuracy of the result, especially in real time scenarios.

Over the course of the last two decades, some algorithms and techniques have been proposed in order to detect whether two or more images are similar. These algorithms

and techniques are a fundamental and essential part of the object matching problem, but still they are not the solution to the problem.

The object matching problem is analysed through a series of steps in order to achieve an accurate match between two images. The first and most essential step in this process cycle is the images' features recognition and storage. The purpose of this kind of technique is to extract distinctive image features from both images and to save them in a database of features. SIFT and Harris corner detection constitutes examples of these algorithms. The next step is to match the extracted features between the two images, and define the putative matches. This step can be implemented by means of a matching technique like the Correlation window method. The last step as well as the most essential of an object matching procedure, is the estimation of the accurate matches and the calculation of the matching rate. The correct matches' estimation can be achieved by applying a sampling algorithm. This type of algorithm is using a mathematical model based on Epipolar geometry or/and Euclidean distance for the calculation of image inliers. A successful algorithm in this field is RANSAC algorithm.

An effective solution to the object matching problem can be produced by incorporating all the introduced algorithms. These will be analysed individually in the following chapters.

1.1 Aim & Objectives

This section describes in detail the aims and the objectives of the project and analyses the structure of this document.

1.1.1 Aim

The primary aim of this project is to design and develop an application that will be able to decide whether two or more images match, calculate matching ratio and determine their common object. The proposed application must be capable of handling images with poor illumination, different scale and rotation and to verify the effectiveness using the RS company electrical components.

1.1.2 Algorithm Objectives

This project has two main objectives. Defining the collaboration between the three algorithms, SIFT, Window Correlation matching and RANSAC algorithms. The selection process of these algorithms can be found in the interim report, in which a comparison between them is included. This collaboration involves the declaration of an execution cycle for each algorithm and the type and number of parameters that will be interchanged within the application, based on the theoretical background of each algorithm.

The second objective and the most complicated one, is to replicate and optimise these algorithms in order to solve any problem and error that may occur during the program

implementation. Algorithms implementation will be executed in MATLAB programming language. MATLAB is a high level programming language which provides development tools which facilitate the rapid development and handling of images. Algorithms optimisation with respect to the matching accuracy and speed are also included.

1.2 Report Structure

This report consists of five main sections. Firstly (Chapter 2), the theoretical background is described in detail. An introduction to digital image matching is described using previous experiments. In addition the requirements of object matching software are explained. Each algorithm structure is scrutinised with respect to the mathematical background, explained in each of the processing steps independently. Also, in this section some results from previous implementations are presented.

In the basic framework and implementation section (Chapter 3), the developed application is explained. Each algorithm and function implementation step is analysed using diagrams, flowcharts and figures in accordance with the project requirements.

In the next section (Chapter 4), application tests and results are presented and examined in terms of matching accuracy and speed. Project application will be tested under some basic and thorough tests. At the end of this section, graphs will be used to verify the matching rate results.

Through the fourth section (Chapter 5), project conclusions will be discussed in relation to the results obtained. In addition, a project evaluation will critically assess the project's achievements and determine whether it has met and achieved its objectives. Finally, in the last section (Chapter 6), future improvements and work are proposed.

The code of each of the main functions can be found in Appendix section 9.

2. Theoretical Background

This chapter is a review of the work undertaken in object matching applications. To this effect, the three main algorithms which are utilised in this project are introduced and analysed on the basis of their theoretical and mathematical background.

2.1 A Review of Previous Work in Object Matching

There are several studies related to the digital image matching field. Important research studies have been carried out during the last five years. Project algorithms, SIFT, Correlation Window matching and RANSAC were previously used, in conjunction with other significant algorithms for object matching processing. The main purpose of an image object matching software is to determine the matching ratio and whether the images match in comparison. With this type of software, it is essential to ascertain the correct common points between the images accurately and not a relation between their points. Previous influential works are briefly described below.

- Harris corner, window correlation and RANSAC paradigms were implemented together for object matching operations. Harris corner algorithm was used in relation to corner points (keypoints) detection. Furthermore, window correlation and RANSAC algorithms were employed in order to identify putative matches and inliers (valid points) respectively. Results demonstrated weaknesses in matching two images when there is a large change in the images' viewpoint. Also, confusion occurred in repetitive textures and some image pairs were computed incorrectly. A satisfactory result was produced, which was not however sufficient when a detailed object matching was required [17].
- SIFT and RANSAC algorithms were also previously used for object matching. This approach was more robust than the previous one and was considered as a good approach where speed is a requirement. However, there is potential for improvements [17].
- An enhanced implementation of RANSAC, coded as MAC-RANSAC, outperforms RANSAC avoiding some of its main limitations and drawbacks, with the exception of the maximal number of iterations which is fixed considering time complexity [16].
- Finally, the *a contrario* model surpasses some of the best matching methods such as M-estimators, LmedS and classical RANSAC by estimating the relative points between two images, even if there is a large number of outliers up to 90% [15].

All of these recent implementations show that there is potential for improvements and optimizations that can be performed to achieve better results with respect to accuracy and application efficiency. Some of the referred models' results are presented below, *see figures 1(a-d)*.

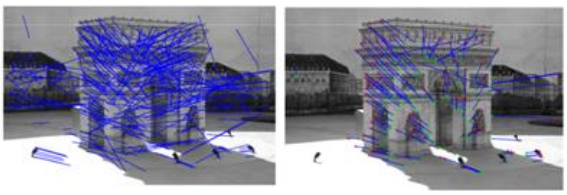

Figure 1(a): Harris Corner + Window Correlation + RANSAC [17].

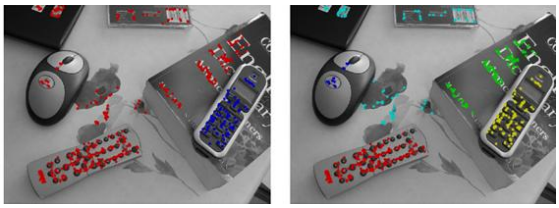

Figure 1(b): MAC-RANSAC algorithm [16].

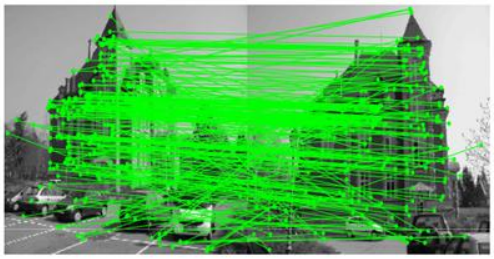

Figure 1(c): RANSAC + SIFT [17].

Figure 1(d): *A contrario* model [17].

2.2 SIFT Feature based Object Recognition

The SIFT algorithm, proposed by David Lowe in 1999, is an image processing algorithm which can be used to identify and extract the distinct features of an image. Once these features have been detected from the two reference images, they can be stored in the database of features, which may be a different file. The SIFT algorithm output is a set of keypoint descriptors and not the result of a two-image comparison. As long as the descriptors have been generated and stored, the image matching procedure can begin image matching. For image matching the nearest neighbour algorithm can be used [1, 2]. The SIFT algorithm processing cycle can be observed in *figure 1*.

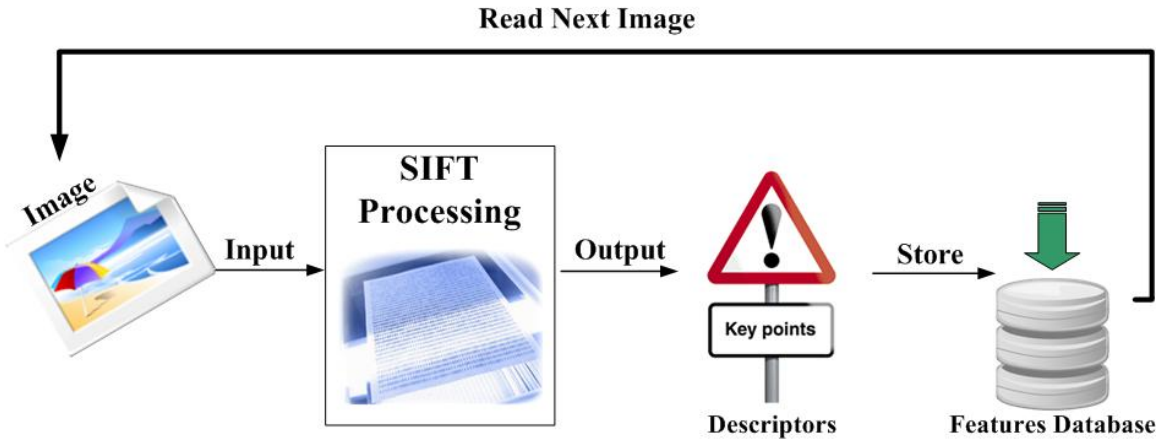

Figure 2: SIFT algorithm processing cycle.

2.2.1 SIFT Algorithm Types of Invariance

The SIFT algorithm should be effective and to operate accurately within all sorts of image irregularities that may occur between images. The SIFT algorithm should be invariant to image [6, 7]:

- Illumination
- Image noise
- Scaling
- Rotation
- Changes in viewpoint

2.2.2 SIFT Algorithm Computational Phases

The SIFT algorithm consists of four phases. These phases simplify the operation whilst facilitating understanding regarding the SIFT processing cycle [1, 2]. The four phases are:

1. Scale-space Extrema Detection
2. Keypoint Localisation
3. Orientation Assignment
4. Keypoint Descriptor

2.2.2.1 Phase 1: Scale-space Extrema Detection

The first computation phase identifies effective interest points. It runs through all scales and image locations and seeks out for identical potential points. The computation is achieved efficiently by using the Difference-of-Gaussian (DoG) function. The resulting potential interest points of the computation are invariant to image scale rotation and orientation [1, 2].

Image scale space is defined as the function $L(x, y, \sigma)$ which is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$ and an input image, $I(x, y)$ [1,7]:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{1}$$

,where * represents the convolution operation for x, y and σ the feature point scale, thus:

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

For the efficient detection of stable keypoint locations in scale space, DoG function convolved with the image, $D(x, y, \sigma)$ is used and is computed from the difference of [1]:

$$D(x,y,\sigma) = \left(G(x,y,k\sigma) - G(x,y,\sigma)\right) * I(x,y) \quad (3)$$

,where k is a constant multiplicative factor.

By employing the above equation it is given that:

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) \quad (4)$$

,which is simply the difference between Gaussian blurred images L, at scales σ and $k\sigma$ [7].

The first step for potential interest points detection is the repeated image convolution with Gaussian filters at different scales, and the generation of a set of scale space images, *see figure 3 (left side)* [7]. Then the subtraction of adjacent Gaussian images (is applied to produce the DoG images *(right side)*. At the time that an each octave has been computed, the processing cycle is over and the Gaussian image is down-sampled by a factor of two (2). Subsequently the processing cycle is repeated again [1].The resulting convolved images are grouped together by an octave (where an octave corresponds to the parameter σ by double). The value of k parameter is selected and as a result of which a fixed number of blurred and DoG images are obtained [1, 7].
It is important to mention an approximation to the scale-normalised Laplacian of Gaussian $\sigma^2\nabla^2 G$, which is provided by DoG filter.

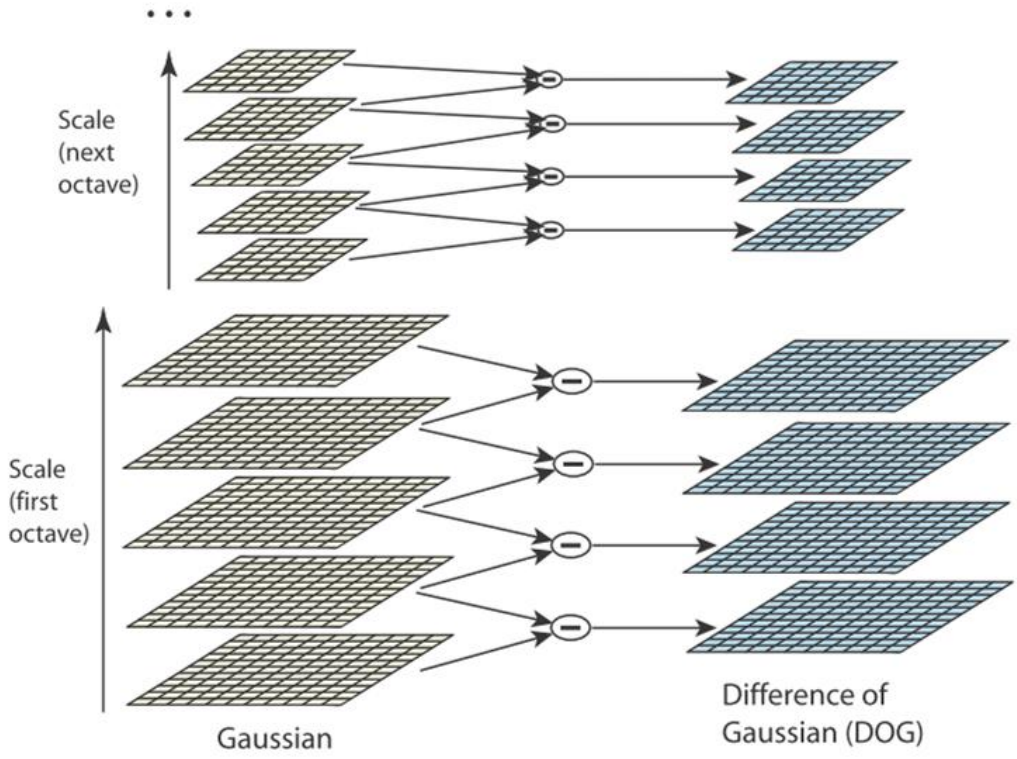

Figure 3: Scale-space Extrema Detection process cycle. Computation of DoG images [1].

The next step within the first phase is the detection of local maxima and minima of the image (keypoints), $D(x, y, \sigma)$, where each sampled point is compared against its nine (9) neighbours in the same image and the 18 neighbours in the scale above and below (nine (9) neighbours in each scale), *see figure 4*. Provided that the pixel is a local maximum or minimum, then it is defined as a candidate keypoint, and is selected only if it is larger or smaller than all of its 26 neighbours together [1, 7].

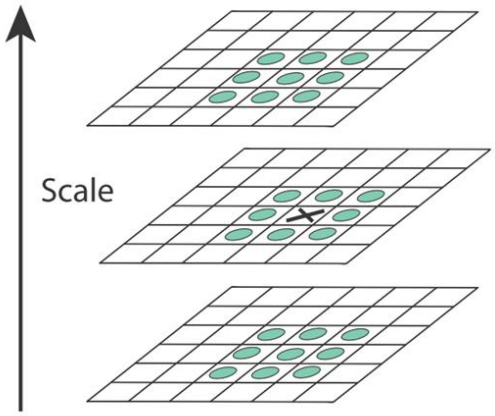

Figure 4: Local maxima and minima detection of the DoG images. Comparing "X" pixel to its 26 neighbours [1].

2.2.2.2 Phase 2: Keypoint Localisation

The second phase of the SIFT algorithm uses all interest potential points that are found in the first phase, and creates a detailed model for location and scale determination. Stability is the keypoint in the selection criterion. Each stable keypoint is a keypoint that is resistant to image distortion [2]. Unstable points are rejected, since it means that they have low contrast and therefore are sensitive to image noise.

Initially, this approach locates keypoints relative to central sample point scale and location. To determine the interpolation of the maximum, a three dimensional (3D) quadratic function is fitted to the local sample points. This provides effectively improves stability matching.

The scale-space function $D(x, y, \sigma)$ is shifted, and thus the origin is at the sample point [1]:

$$D(\mathrm{x}) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \tag{5}$$

,where D and its derivatives are estimated at the sample point and $x = (x, y, \sigma)^T$ is the offset from this point.

By setting the derivative with respect to x to zero, the location of extremum can be determined [1]:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{6}$$

,where $\hat{x}$ is the location of the extremum and if the offset $\hat{x}$ is greater than 0.5 in any dimension, this means that the extremum comes closer to a different point.

In this case, the sample point will change and the interpolation will be performed. In order to obtain the interpolation estimation for the extremum location, the final offset $\hat{x}$ and the sample point are added.

The return value of extremum function $D(\hat{x})$ proves useful in order to reject unstable extrema with low contrast. This function can be obtained by substituting the two previous equations and thus giving:

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{7}$$

In *figures 5a-5d*, the effect of keypoint selection on a natural image can be observed. These examples are experiments obtained by [1] paper, in relation to all of which extrema with value of $|D(\hat{x})|$ less than 0.03 were rejected, *see figures 5(a-c)*. The four examples provided below are helpful to understand how the keypoint selection is performed, and how this selection process can be improved.

A 233x189 pixel image is used. A low resolution image was selected in order to avoid increased cluster amount. Keypoints are present in the image as vectors determine the location, scale (arrow length) and orientation (arrow orientation) for each keypoint.

Figure 5(a): Original image with reduced contrast.

Figure 5(b): Detection of 832 keypoints at maxima and minima of DoG function.

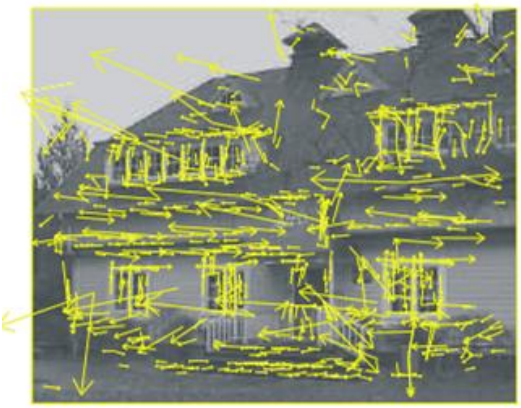

Figure 5(c): Keypoints reduced to 729, due to $|D(\hat{x})|$ =0.03 factor.

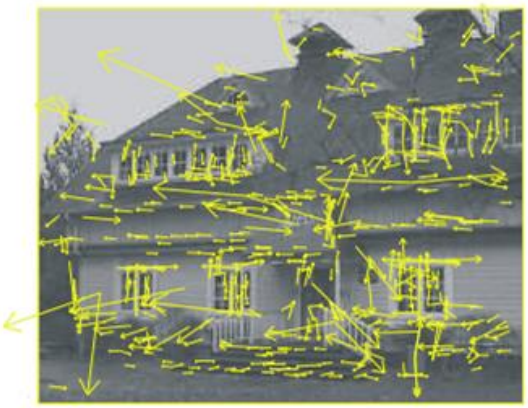

Figure 5(d): Factor *r=10*. Keypoints with ratio greater than 10 are eliminated.

Eliminating edge responses serve to increase stability performance. Rejecting low contrast keypoints is not enough. The DoG function has a strong response along the edges; therefore unstable keypoints which are susceptible to small amounts of noise are detected. For each poor edge found in DoG function, a principal curvature will appear across the edge but also a small one in the perpendicular direction. By using a 2x2 Hessian matrix, the principal curvatures can be computed at keypoint location and scale [1]:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{8}$$

,where **H** is a Hessian matrix.

By taking the differences of neighbouring sample points, the derivatives can be computed. **H** matrix eigenvalues are proportional to the principal curvatures of *D*. Since we are concerned only with the eigenvalues ratio, their computation can be avoided. The next step is the computation of the eigenvalues' sum from the trace of **H** matrix as well as their product from the determinant [1]:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \,, \tag{9}$$

$$Det(H) = D_{xx}D_{yy} - (Dxy)^2 = \alpha\beta \tag{10}$$

where α is the eigenvalue with the largest magnitude and β with the smallest one.

In the computation of the determinant there is a possibility for the result to be negative, thus the point is discarded since it is not considered as extremum given that the curvatures have different signs. Let r be the ratio between the smaller and the largest eigenvalues, thus, $\alpha = r\beta$, which results in the equation shown below [1]:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{11}$$

Observing the above simplification we can take notice of the fact that the new point depends only on the eigenvalues' ratio rather than on their individual values. The $(r+1)^2/r$ obtains its minimum value when the two eigenvalues are equal and it increases with r. Therefore, if we want to perform a check whether the ratio of the principal curvatures is below a defined threshold r, we only need to check whether the following equation is satisfied [1]:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{12}$$

This can be computed efficiently, when operations with less than 20 floating point are required. In the example provided, the r value is set to ten (10), which means that the keypoints with ratio greater than ten (10) between the principal curvatures are rejected. The effect of this operation can be observed in *figure 5d* [1].

2.2.2.3 Phase 3: Orientation Assignment

In the third phase, [2] SIFT computes the gradient of the keypoints identified in the second phase. For each keypoint, one or more orientations are assigned on the basis of local image gradient directions. Features operations that may be performed in image data will be transformed relative to each feature's assigned orientation, scale and location, thus providing invariance in these transformations [1].

For each candidate keypoint the following rules are applied [7]:

- Keypoint position is determined by the interpolation of its closed data.
- Low contrast keypoints are rejected.
- Determined keypoint orientation.
- Responses along edges are rejected.

The Gaussian blurred image L with the closest scale is selected comparatively to the keypoint scale. To this effect, all computations are performed in a scale-invariant manner. Subsequently, for each image sample, $L(x,y)$, on that scale, the gradient magnitude $m(x,y)$ and the orientation $\theta(x,y)$ is pre-computed using pixel differences[1]:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 - (L(x,y+1) - L(x,y-1))^2} \tag{13}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{14}$$

Keypoint orientation is determined by computing the gradient orientation histogram of the keypoint neighbourhood. This is computed using the Gauissian image at the scale which is closest to the keypoint's scale. Each neighbouring pixel is weighted by the

gradient magnitude and by a Gaussian at a scale of 1.5 times of the keypoint (σ =1.5 times).

Dominant orientations are defined by observing the computed histogram peaks. A new identical keypoint is generated for the corresponding direction to the histogram maximum and for any direction which is greater than 80% of the maximum value. All of the keypoint properties are computed comparatively to the keypoint orientation. This feature provides invariance to rotation [7, 1].

2.2.2.4 Phase 4: Keypoint Descriptor

The previous SIFT phases are assigned to each selected keypoint information, such as image location, scale and orientation. A two dimensional (2D) coordinate system is used to represent the local image region which must have invariance to these parameters [1].

The final phase computes a descriptor for each local image sample point in a region around the keypoint location [2, 1]. The keypoints' descriptors are highly distinctive but should be as invariant as possible to the remaining variations, such as changes in three dimensional (3D) viewpoint plan and illumination [1].

For each keypoint, the magnitude of local image gradients together with orientation are measured at the selected scale in that region, *see figure 6 (left side),* and these keypoints are converted into a representation which allows for sensible levels of local shape change in illumination and distortion.

A weight is applied to each of the samples. This weight has been assigned by a Gaussian window with σ =1.5 times the width of the descriptor window, defined by the overlaid circle *(left side)*. The Gaussian window method is applied to avoid any sudden descriptor changes [1, 7].

The weighted samples are accumulated into orientation histograms with the objective of summarising their contents into 4x4 sub-regions *(right side).* Figure 6 shows six (6) directions whereby the length of each arrow represents the sum of the gradient magnitudes near that direction within the specified region. An 8x8 set of samples is used to compute a 2x2 descriptor array.

Changes to image contrast will be cancelled by vector normalization, since both image pixel values and gradient will be multiplied by the same constant. Variations in brightness may also occur, but which result to no changes given that a constant value (due to brightness) is added in relation to all image pixels. The reason behind this is because gradient values are not affected in view of the fact that they are computed by pixel differences resulting in a descriptor, which is invariant to affine changes in illumination [1].

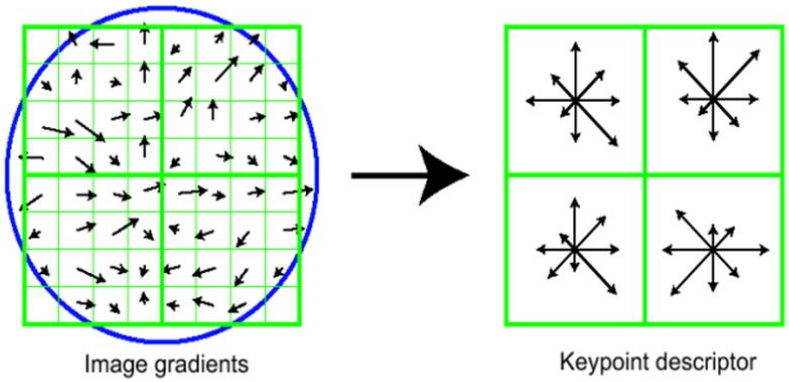

Figure 6: SIFT feature descriptor [1].

2.2.3 SIFT & Image Matching

The SIFT algorithm output is a set of distinctive keypoint descriptors and not the result of a two-image comparison. Once the keypoint descriptors set has generated from each image, the image matching processing can begin. Image matching processing can be performed by using a Matching Algorithm, *see figure 2*. Various implementations can be employed to carry out image matching, each one with different characteristics [2].

As mentioned in the previous sub-section, keypoint descriptors are highly distinctive. A descriptor feature can be matched with a feature within the features database, whereby there is good probability for such matching. Nevertheless, in the case of a cluttered image, many background features will not match correctly. Therefore, many false matches will be predicted in addition to the correct ones. By identifying keypoint subsets that match object parameters (location, scale and orientation), accurate matches can be filtered from the full set of matches. The probability that an individual feature match will be an erroneous one is higher than the probability that a series of features will match these parameters.

A solution to this problem can be achieved with the use of an efficient keypoints distance estimation algorithm, with the use of which it is decided whether a dataset is matched based on a mathematical model for the determination of these accurate clusters. Such algorithm is the RANSAC which will be described in detail in the next section.

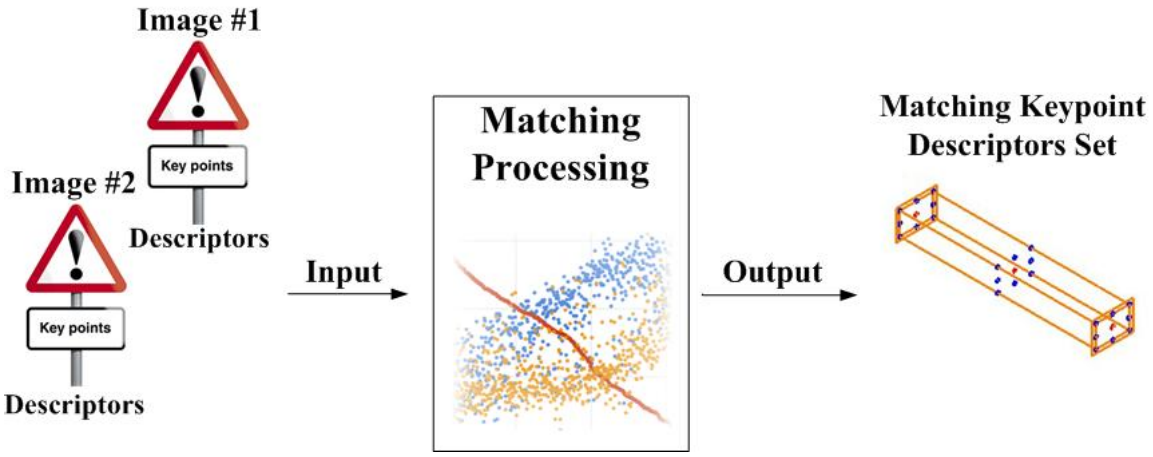

Figure 7: SIFT descriptors matching procedure. Two images keypoint descriptors are the input to matching algorithm and the output is a set of matched similar keypoint descriptors.

2.3 Cross - correlation Window Matching

2.3.1 Correlation Method

The Correlation is a statistical method that is utilised for the comparison of two variables. It consists of a quantitative measurement of data within a data set. Correlation method measures the relation between two variables (X and Y) and quantitates the strength of this relation. The strength is determined by the linear relationship between the X and Y variables. This relationship is represented by the number of points that are found to be fixed within a straight line [20].

The result of the correlation between X and Y variables is the Pearson product-moment correlation coefficient, which has been originally proposed by Francis Galton in 1886 and subsequently formalized by Karl Pearson in 1896 [19]. Pearson correlation coefficient quantifies the relationship between the X and Y variables which is denoted with the letter *r*. When all the points within a scatter plot line have an upward incline, *r* is equal to plus one (+1). On the other hand, when the line has downward direction, *r* is equal to minus one (-1). The *r* values which are closer to plus one (+1) and minus one (-1) indicate a strong correlation. When the outcome is equal to plus one (+1) it means that the images are identical. In contrast, the minus one (-1) indicate same image but flipped around. The values which are closer to zero (0) denote a weak correlation between X and Y variables whilst a zero (0) value indicates that the two variables have nothing in common [19, 20]. Examples of correlation can be observed in scatter plot graphs in *figures 8(a-d)*.

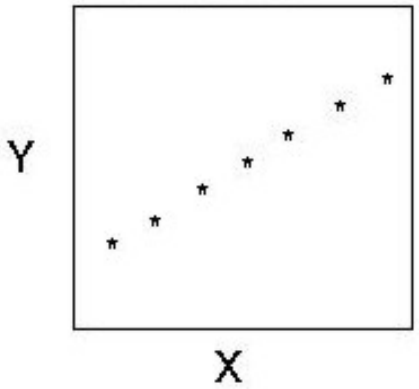

Figure 8(a): Perfect Positive Correlation r = 1

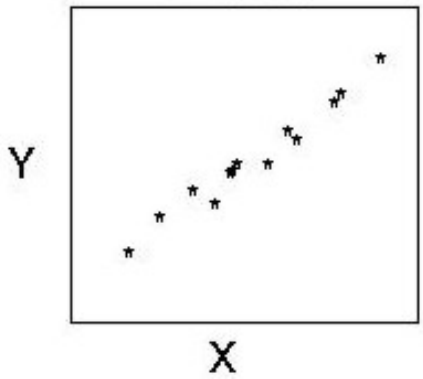

Figure 8(b): Strong Positive Correlation, r >= 0.8

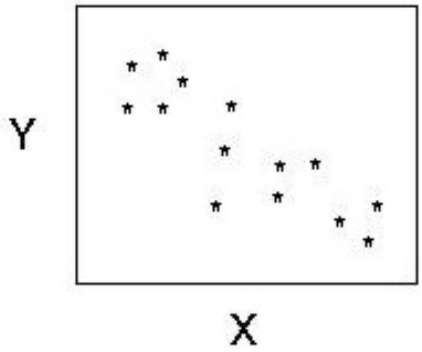

Figure 8(c): Weak Negative Correlation, r<=-0.5

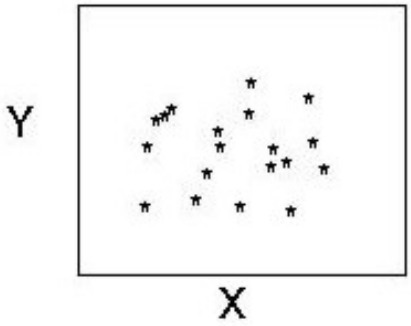

Figure 8(d): Zero Correlation, r=0

2.3.2 Pearson's Correlation Coefficient

As mentioned in the previous section, the result of the correlation is the correlation coefficient *r*, which takes values between plus one (+1) and minus one (-1). This value reflects the amount of commonality found between the two variables [19].

To calculate the correlation coefficient, the sum of squares (ΣΣ) for each variable X and Y plus the sum of the cross products of X and Y is required [19, 22].

The sum of squares for X variables is:

$$\Sigma\Sigma_{XX} = \sum (Xi - \overline{X})^2 \tag{15}$$

The sum of squares for Y variables is:

$$\Sigma\Sigma_{YY} = \sum (Yi - \overline{Y})^2 \tag{16}$$

The sum of the cross-products of X and Y is:

$$\Sigma\Sigma_{XY} = \sum (Xi - \overline{X}).(Yi - \overline{Y}) \tag{17}$$

,where $\overline{X}$ and $\overline{Y}$ is the mean for variable X and Y respectively.

By using the above equations, correlation coefficient *r* can be calculated as shown below [19, 22, 23]:

$$r = \frac{\Sigma\Sigma_{XY}}{\sqrt{(\Sigma\Sigma_{XX})(\Sigma\Sigma_{YY})}} \tag{18}$$

2.3.3 Cross correlation

Cross correlation is an image matching technique, which employs basic correlation approach in order to match different portions of images against one another. Cross correlation is applicable for the purposes of image processing because it uses a pixel-by-pixel comparison for an area of interest within the image under analysis. The result of cross correlation is an image with a set of spots, where each spot shows the position of that image which matches to the reference image. Each spot is described by a grayscale value, where the brighter spot represents a better match. To eliminate false matches, a threshold value can be set to determine whether a spot represents an inlier or an outlier. Values below that threshold will be discarded [24]. The image area under interest, can be represented as a square window (i.e. 5x5 pixels for a 250x300 image), which it is compared with the pixels in the reference image.

Cross correlation operation is similar to the operation relating to the spatial convolution one, where two functions are combined to form a third one [27]. In cross correlation, the image mask, which is the correlation window, is moved pixel-by-pixel to the input image and a third image is used for placing the result [24]. This process is illustrated in *figure 9*.

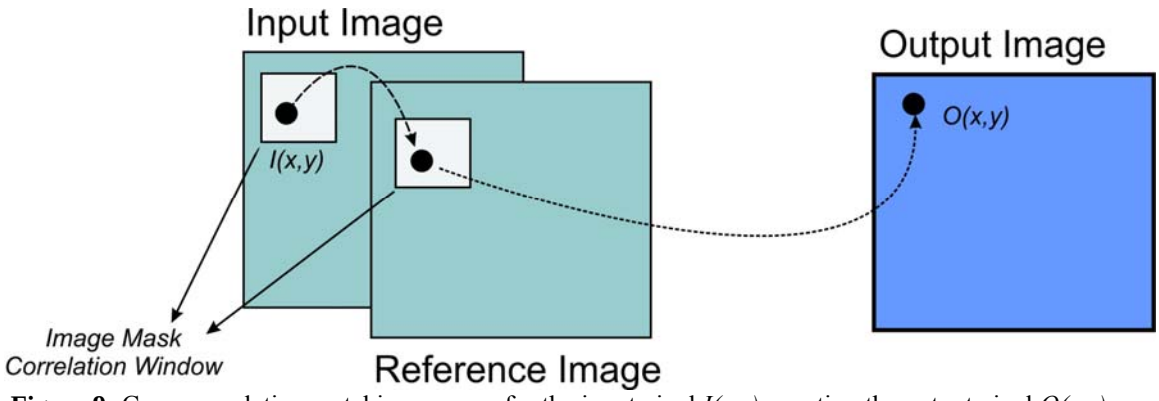

Figure 9: Cross correlation matching process for the input pixel *I(x,y)*, creating the output pixel *O(x,y)*.

The results of the cross correlation, when applied to two dimensional (2D) images, represent an array of points called the correlation map. The correlation map defines the common points between the two images. Where there is no correlation between the two

images, the output values will be relatively small. These values can be significantly large where the brightest values tend to be equal. The basic function of cross correlation for template matching is influenced by the square Euclidean distance measure and is defined as follows [24, 26]:

$$d^2_{I,k}(i,j) = \sum_{x,y} \left[I(x,y) - k(x-i, y-j) \right]^2 \tag{19}$$

,where *I(x, y)* is the input image, with the sum over *x* and *y* containing the feature point *k* at (*i, j)* position.

By simplifying the above equation:

$$d^2_{I,k}(i,j) = \sum_{x,y} \left[I^2(x,y) - 2I(x,y)k(x-i, y-j) + k^2(x-i, y-j) \right] \tag{20}$$

,where $\sum k^2(x-i, y-j)$ is a constant. Therefore, by assuming that $\sum I^2(x,y)$ is constant:

$$O(x,y) = \sum_{x,y} I(x,y) * k(x-i, y-j), \tag{21}$$

or

$$O(x,y) = \sum_{i,j} I(x+i, y+j) * k(i,j) \tag{22}$$

,where *O(x, y)* is the output of the cross correlation function and which describes the similarity between the image and the feature [26].

The aforementioned technique faces a significant limitation. On the occasions where the image $\sum I(x,y)$ activity varies according to the position, then the use of the above equation is not suggested [26]. Despite this drawback, it can be employed successfully when the input image will potentially match exactly or when it is close to the feature point [24].

The above limitation aside, the correlation coefficient technique (18) overcomes these problems since the image and feature vectors are normalised to unit length. Although the correlation coefficient technique provides a solution in relation to the cross correlation problems, there is no feedback as to how accurately it corresponds when a large rotation is applied between the two images.

2.4 RANSAC Keypoints Estimation Algorithm

The RANSAC algorithm was proposed by Fischler and Bolles in 1981. RANSAC is a very successful and robust estimator algorithm which gives a solution to image matching problems [10]. It was introduced initially as a method or technique to estimate the parameters of an input model, and designed in such way to cope with a large number of outliers in the input data [11, 12].

The RANSAC algorithm relies on a re-sampling technique which can generate entrant solutions. This is achieved by using the minimum number of observations required for the input model parameters determination. The RANSAC algorithm starts by bringing into play the smallest possible set and through its operation the set is enlarged with consistent data points [12].

2.4.1 RANSAC Basic Operation

RANSAC algorithm processing is simple and effective. Initially, two points (keypoints) that define a line are selected randomly. The "correctness" of this line depends on the number of image points that occur within a distance threshold *t*. This random selection is repeated for *n* number of times and the line with the most accurate results is considered as being the most robust one. The points inside the specified threshold distance are the inliers (Consensus Set). In a case when at least one line point is considered to constitute an outliner, the line's "correctness" will be controverted, *see figures 7a-7e*. If more than one line is found which is considered as being "correct" (only accurate points exist and no outliers), then the line with the most "correctness" will be selected. If line A has a "correctness" equal to nine (9) and a line B has a "correctness" equal to six (6), then line A will be selected. This example is refers to a one dimensional (1D) transformation, thus, two points are enough to determine the model [10].

In image processing, the problem solution is more complicated since the model estimates a planar homography. The minimal subset of a set of two dimensional (2D) point correspondences consists of four correspondences, given that now each point in an image is represented in X, Y coordinates. Consider an image as a graph where points A and B are represented by (x, y) and (z, e), respectively [10]. *See figures 10 (a-e)* [18].

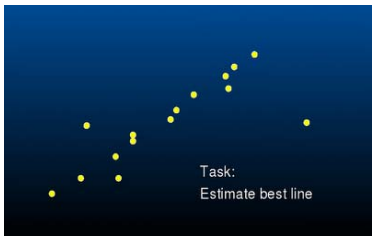

Figure 10(a): Task 1 – Line estimation.

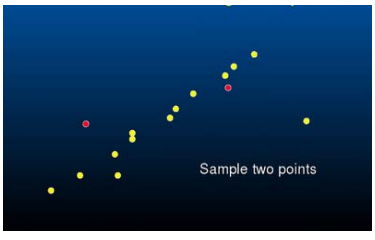

Figure 10(b): Task 2 – Two random points are selected.

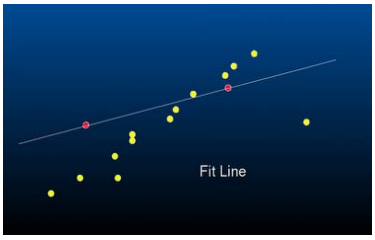

Figure 10(c): Task 3 – A line is fitted.

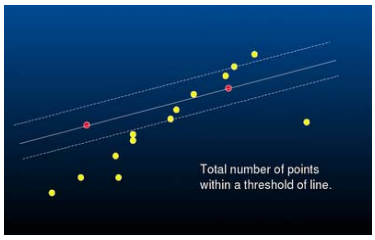

Figure 10(d): Task 4 – Count number of points with a line.

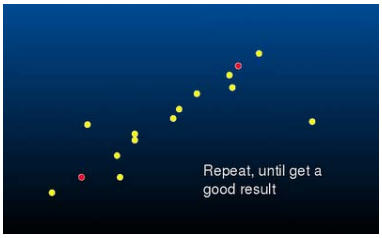

Figure 10(e): Task 5 – Repeat process until line with most high "correctness" is found.

2.4.2 RANSAC Computational Phases

The RANSAC algorithm is composed of two indispensable phases that are repeated until the predefined problem requierments have been satisfied. These two steps are mentioned below and will be analysed individually in the following sub-sections [11].

1. Hypothesize
2. Consistency Test

RANSAC does not stop neither at the end of the second phase nor on a predefined number of iterations. It terminates when the probability of finding a better ranked Consensus Set falls below the predefined threshold value.

2.4.2.1 Phase 1: Hypothesise

The first phase of RANSAC's operation involves the random selection of the Minimal Sample Sets (MMSs) from the input data set. It is important to mention that image extracted features compromise the input data set, where the RANSAC algorithm is not involved in this operation. Since MMSs are randomly selected, MMSs elements are used to compute model parameters. MMSs cardinality is small enough to specify the model parameters [10, 11].

2.4.2.2 Phase 2: Consistency Test

In the second phase of the algorithm, the entire dataset elements are examined for their consistency relative to the model, which was initialised with the computed parameters of the first phase [10, 11].

2.4.3 RANSAC Algorithm Processing Cycle

The RANSAC algorithm process cycle can be split into five steps. These five steps facilitate towards the algorithm' comprehension and simplify the implementation process by breaking it up into smaller and easier to design procedures.

2.4.3.1 RANSAC Five Process Steps & Parameters Computation

To facilitate the subsequent discussion, it is useful and convenient to introduce a suitable and common representation of the following five steps. *S* represents the input data set, *s* is the number of data points, *T* the threshold, *t* is model threshold and *N* the number of trials.

The five algorithm process steps are [10, 12] listed below and a Data Flowchart Diagram (DFD) of the algorithm can be observed in *figure 8*.

(1) Select randomly a number *s* of data points (distinct keypoints) from *S* and then instantiate the model from this subset.

(2) Determine the data points, inliers S_i (CS), from *S* data set. Inliers should be within the distance threshold *t*.

(3) If the inliers S_i number fraction is greater than a defined threshold *T*, re-estimate the model parameters using the corresponding inliers and afterwards terminate.

(4) On the occasion that $S_i > T$, select a new subset and repeat steps 1 to 3.

(5) After *N* number of trials, the S_i (CS), with the best ranking is selected. Then, the model is re-estimated using the corresponding inliers in the S_i subset and the process is terminated.

The RANSAC algorithm has three undefined parameters; each of them has a different computation mechanism that leads to its identification. The unspecified parameters are mentioned and explored below [9]:

- Error tolerance, defined as *e.*
- The number of attempt subsets, *N.*
- Distance threshold, *t.*

It is computationally unnecessary and inefficient to check every possible sample. Thus, the number of iterations *N* is chosen to be large enough, with a probability *p*, that at least one of the random data points *s*, is free from outliers. Probability *p* is usually set to 0.99, so as to minimise the probability where there is no consistency. Let *w* be the probability where any selected data point is an inlier, and *e* the probability that it is an outlier.

Therefore, $e = 1 - w$. By using this and assuming that N is the number of iterations required for a sample of s data points it means that [9, 10]:

$$1 - p = (1 - w^s)^N \tag{23}$$

Thereby after some manipulations,

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)} \tag{24}$$

In light of the rule of thumb, the algorithm processing is terminated when the size of the consensus set is similar to the number of inliers that are considered within the data set, taking into account the assumed outliers ratio. For n number of data points, $T = (1-e)n$ [10].The distance threshold value t, is employed to determine whether a sample data (datum) fits into a model and it represents the number of compatible data points used to denote that the correct model has been found. The reason why threshold t value must be large enough, is that it has to satisfy two purposes. The first purpose is to ensure that the correct model for the data was identified. The second one is to satisfy the needs of the final smoothing procedure by ensuring that a sufficient number of common consistent points have been located [9]. In practice, the threshold value is chosen empirically. However, if we assume that α is the probability when a sample data is an inlier, and presuming that the error tolerance is Gaussian with a mean equal to zero (0) and standard deviation σ, and then a value for threshold t can be computed. In such a case, $d^2\perp$, which is the square of the point distance, is a sum of Gaussian variables which follows distribution, where m represent degrees of freedom and is equal to the co-dimension of the exact model. Since the model is a point, the co-dimension is equal to two (2) and the $d^2\perp$ is the sum of squared x, y measurement errors. From the cumulative distribution it is given that [10]:

$$\begin{cases} inlier & d^2\perp < t^2 \\ outlier & d^2\perp \geq t^2 \end{cases}, \text{ with } t^2 = F_m^{-1}(a)\sigma^2 \tag{25}$$

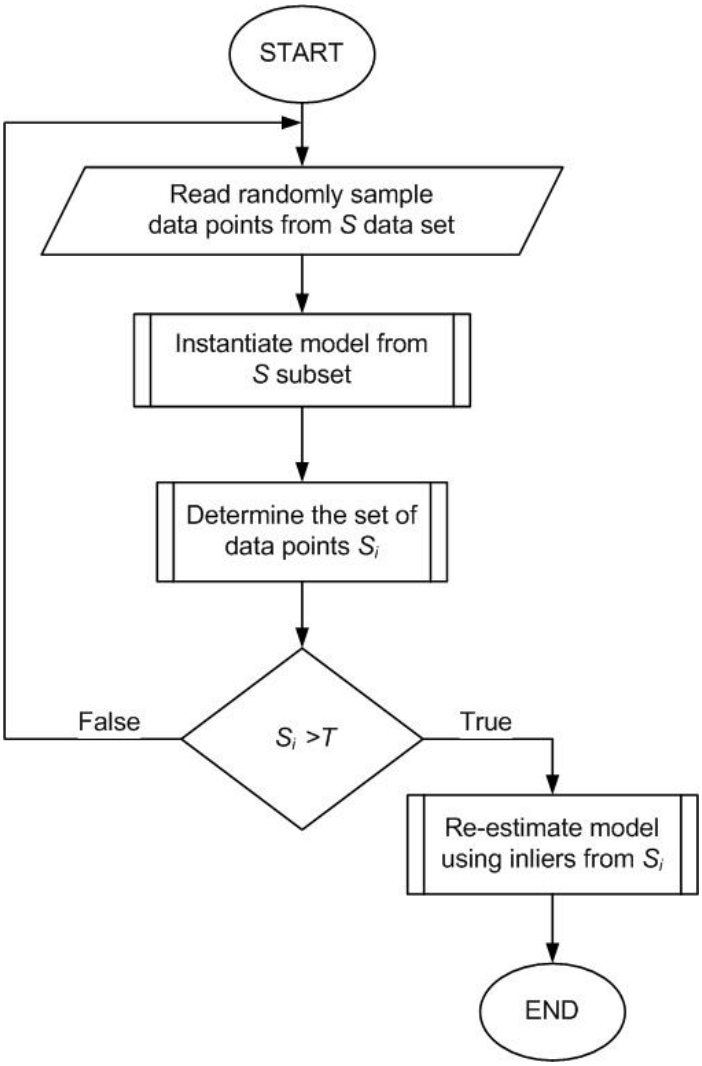

Figure 11: DFD of RANSAC algorithm in five steps process.

2.4.4 RANSAC Errors Solution Mechanism

During RANSAC algorithm operation some errors may occur. These errors derive from local feature detectors and can be separated in two types as follows [9]:

1. **Classification Errors:** This type of error occurs when a keypoint of an image is incorrectly identified as a constitution of a feature. Classification errors are gross errors and have a significant impact and cannot average out. For this reason, the best solution is to select a reliable local feature detector that will collaborate with RANSAC.

2. **Measurement Errors:** This type of error occurs when a feature of an image is correctly identified, but one of its parameters is a slightly miscalculated. Usually measurement errors follow a normal distribution and therefore the smoothing assumption is still applicable to them.

3. Basic Framework & Implementation

As mentioned in the previous section, the proposed application was implemented in MATLAB R2010a. This section analyses and explores the implementation of each algorithm including any problems that have been overcome. A reference to the application structure and how the results are calculated is also included, as well as any other methods tried and tested during the implementation step.

3.1 Application Operation Structure

The three algorithms which are used have already been introduced in the previous sections. The proposed method must employ rotation and scale invariance, extraction and detection of distinctive features from images, without any problems. Application operation structure is illustrated in *figure 12* and described below:

1. **When application starts:** Two images are used as the software input. The first image is the image that a client sends to the company's online system and it could be an image with low illumination and contras. The second one is an image taken from the company database, which is a "perfect" image with sufficient illumination and high contrast. The "perfect" terminology will be explained in the SIFT section. For convenience purposes and lack of time, a vector of images is utilized instead of creating a connection to an SQL database. Both images are converted to grayscale and the image size is reduced if the image is large. This reduction is set by a threshold value, which is under programmer choice. During application testing, the size of both images is approximately reduced to 500x300, depending on image orientation. Both image transformations are used for saving processing overheat. A blur filter is applied in the image for image smoothing, and image contrast is enhanced by subtracting the smooth image by the image minimum value. Then is divided by the image maximum value *see figures 13(a-c)* for images transformation and *figures 14a-14c* for a representation of the images' histograms.

2. **SIFT distinctive feature recognition:** Once the images' transformation is completed, SIFT operation is ready to begin. The SIFT algorithm is implemented as a separate function, where its input consists of the two images and a set of parameters. The result is a vector (database) of distinctive features and a vector of descriptors for each extracted image feature. There is no relation between the two vectors and each one can have different size. As soon as SIFT operation is completed, the two images are printed including the extracted features of each image.

3. **Match by Correlation:** With the end of SIFT operation, each image features vector is used as the input for the match by the correlation function. This function is responsible for the determination of putative matches. The putative matches can be considered as the theoretical matches; this means that with calculated

4. **RANSAC estimation:** The last part of the software is split in two parts.

    a. The first part involves the estimation of the correct matches with the use of the RANSAC mathematical model, which is responsible for estimating and deciding whether a point is correct or not, based on points distance. The RANSAC algorithm is implemented in a different function and its input is a vector, including the inliers between the two images. Once this estimation is completed, a new figure is printed, including the two images common points.

    b. The second part deals with the calculation of the matching rate between the two images. Matching rate is obtained in percentage value, and it is necessary for the user to identify whether two images have a common object or not. More details about matching rate calculation and how software decide if two images have common objects will be explained later, at the end of this chapter.

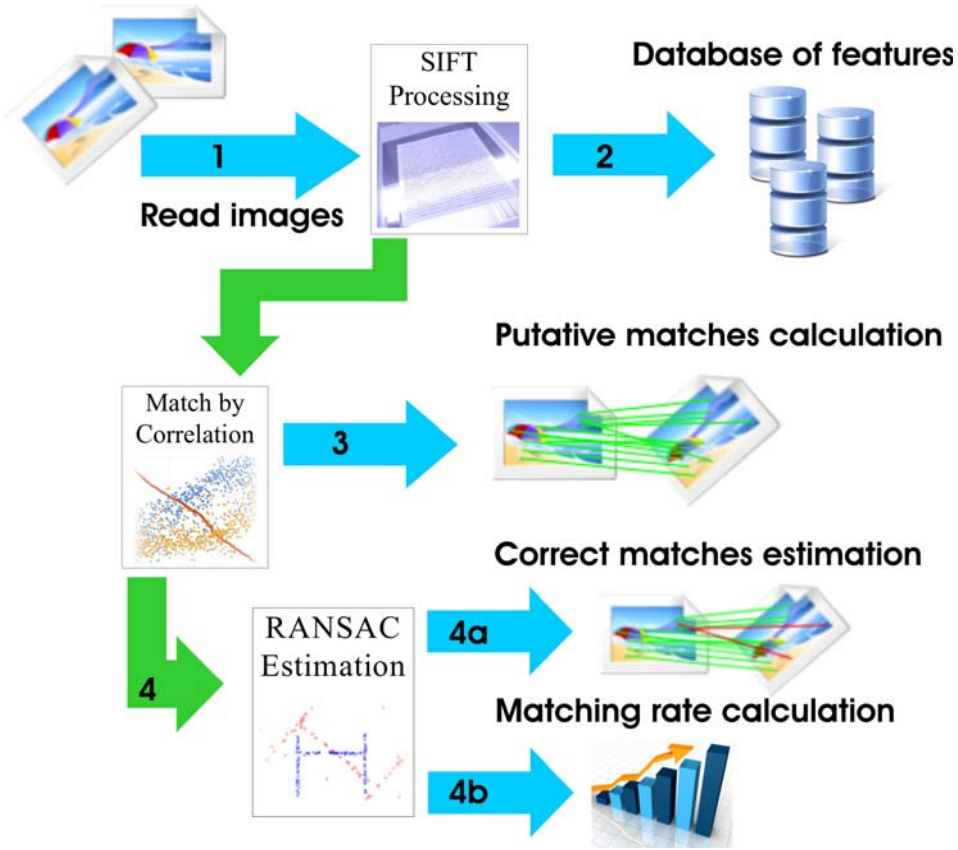

Figure 12: Application operation diagram. For steps numbering see section 3.1 parts 1 to 4.

IMAGE PROCESSING

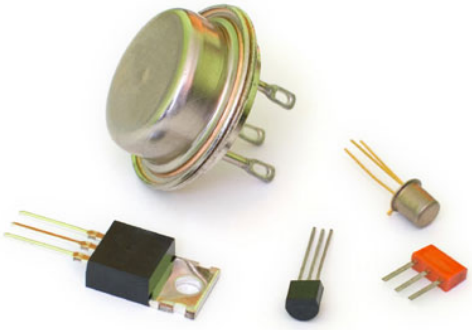

Figure 13a: Input image with size of 400x300 pixels. This is the original image.

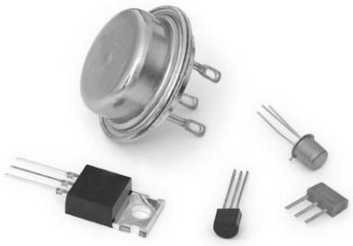

Figure 13b: Image 1 after smoothing.

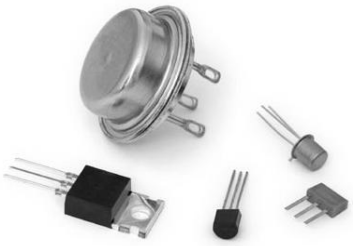

Figure 13c: Image 1 after contrast enhancement.

PROCESSED IMAGE HISTOGRAM

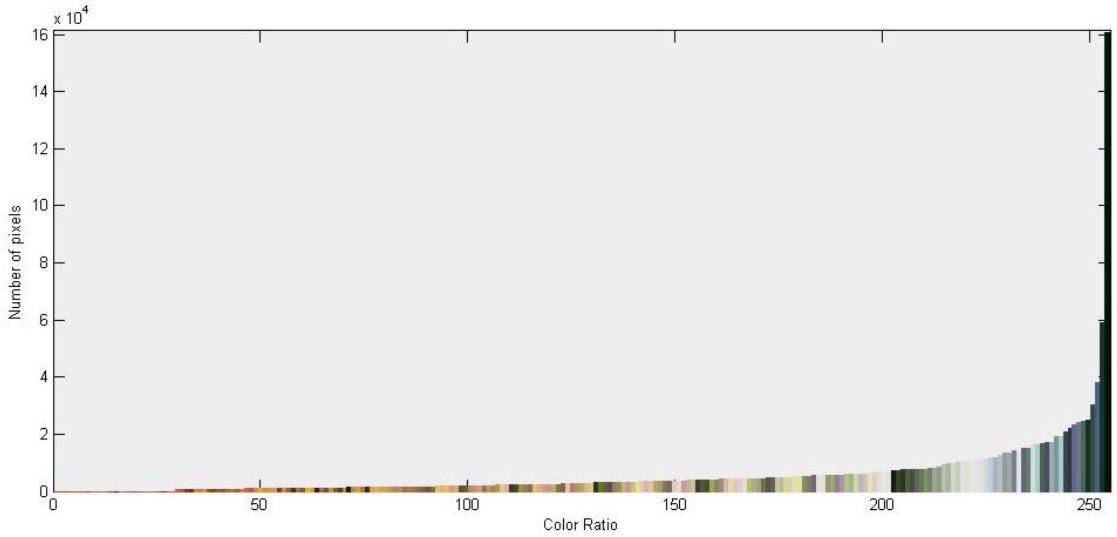

Figure 14a: Colorful image histogram. This is the original image histogram.

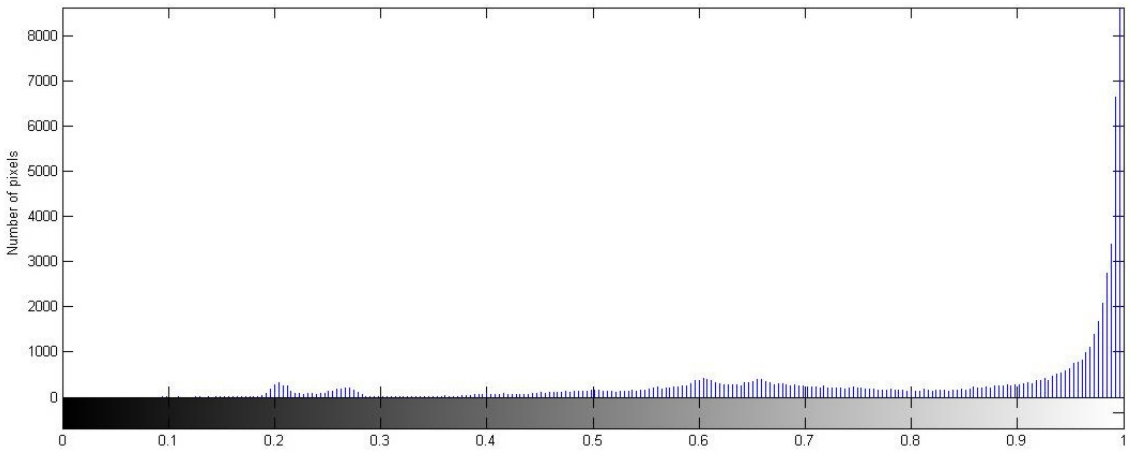

Figure 14b: Grayscale image histogram after smoothing.

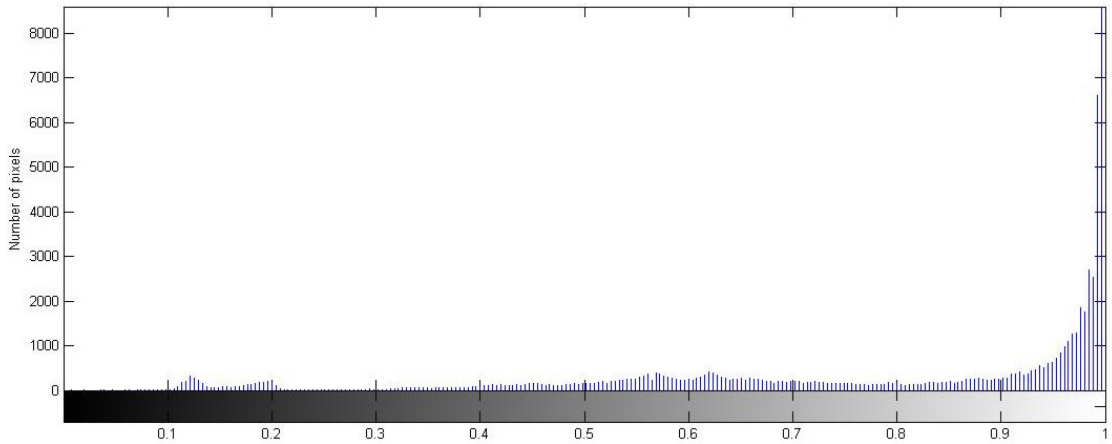

Figure 14c: Grayscale image histogram after contrast enhancement.

3.2 SIFT Feature based Object Recognition [34]

This section describes the SIFT algorithm implementation. As aforementioned for the algorithms' design MATLAB R2010a was employed. The SIFT algorithm is implemented as a separate function and is designed to provide flexibility to the application's user. This flexibility is achieved by offering a wide list of input parameters, which can be set by the user, according to his preference. The default values are equivalent with those chosen by D.G. Lowe. Thus, results which are similar to D.G. Lowe's SIFT version are obtained [1].

The SIFT algorithm it is consist of two main instances, these being the SIFT detector and descriptor.

- The SIFT feature's detector is used to extract image frames and keypoints. These frames and keypoints are oriented points attached to the images in a blob-alike form. The frame tracks these blobs, while the image rotates, translates and scales,

- The SIFT descriptor is a common image edge description that can be found within a frame. Moreover, the generated image descriptors are invariant to image rotation, translation and scaling and can also be robust when there are small image distortions.

The complete processing operation of SIFT function is illustrated in the data flow chart (DFD) diagram shown in *figure 15*.

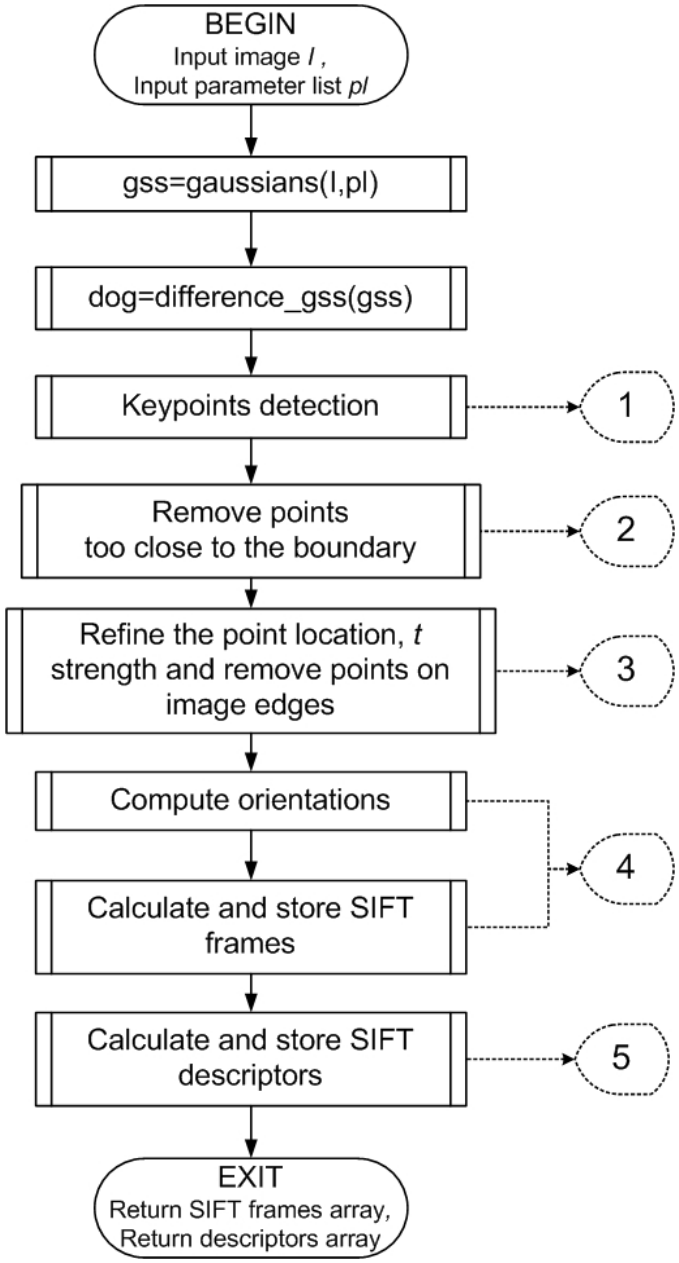

Figure 15: DFD of SIFT algorithm function.

3.2.1 SIFT Function Input Parameters and Return Values

Input Parameters:

As mentioned above, the SIFT function provides a long list of input parameters. This list is informally separated in three categories. The parameters that describe the Gaussian Scale Space, the SIFT detector and the SIFT descriptor are listed as follows:

1. **Gaussian scale space parameters:**

    a. The number of octaves *O*. The default value is thus set to cover all possible feature sizes.

    b. The first octave index, $o_{\min}$. The default value is set to minus one (-1), as in D.G. Lowe's implementation. By setting the default value to minus one (-1), the image pixel values before the Gss are doubled.

    c. The number of sub-levels *S* of each octave. D. G. Lowe's implementation uses *S=3*.

    d. The base smoothing σ_0, which is the first octave zero (0) level smoothing. D. G. Lowe's implementation employs $\sigma_0 = 1.6 \cdot 2^{1/S}$.

    e. The nominal smoothing level σ_n, for the input image.

2. **Detector parameters:**

    a. DoG scale space maxima threshold *t*. Values below that threshold are ignored and smaller values accept more features.

    b. The edge threshold, or in other words the local extrema localisation threshold. If a feature value is below this threshold, it is considered as unstable, and is subsequently ignored. Bigger values accept more features.

    c. The remove boundary points parameter which is responsible for discarding the frames which are found near the image boundary point.

3. **Descriptor parameters:**

    a. The magnification factor *m*. Each spatial bin has $m\sigma$ size and by changing the *m* value the spatial bin size is changed.

    b. The number of orientation bins.

c. The number of spatial bins. This parameter defines the descriptors vector size, where it is equal to *(Spatial Bins number)^2 x (Orientation bins number)*, with radius of *(Spatial Bins number)^2 x mσ/2*.

Returned Values:

The SIFT function returns input image frames and descriptors. Both image frames and descriptors are vectors and are described as follows:

1. Image SIFT frames vector is a 4xN matrix, which is storing one frame per column with the following format:

    a. FRAMES(1:2,n) describes the center of frame n in the position (x, y).
    b. FRAMES(3,n) describes the scale σ of the frame n.
    c. FRAMES(4,n) describe the orientation θ of the frame n.

2. Image SIFT descriptors is a KxN matrix and the default size of K=128.

3.2.2 SIFT Scale Spaces

Both SIFT detector and descriptor are designed based on the Gaussian Scale Space (GSS) of the input image *I(x)*. The Gss scale space is described below and it is designed according to equations 1 and 2:

$$G(x,\sigma) \stackrel{\Delta}{=} (g_\sigma * I)(x) \tag{26}$$

,where x is the spatial coordinate and σ the feature scale coordinate. The g_σ is the Gaussian kernel isotropic of $\sigma^2 I$ variance.

The Gaussian scale space is computed in a function within the SIFT one, and is executed as DFD, shown in *figure 15*. The image has already been smoothed at a nominal level σ_n prior to entering the Gaussian function, as already described in section 3.1 (part 1), and as a result:

$$G(x,\sigma) = (g_{\sqrt{\sigma^2-\sigma_n^\sigma}} I)(x) \tag{27}$$

The octaves pyramid is computed from the bottom incrementally. This is undertaken by successive convolutions which have small kernels. See table 1.

Gaussian Scale Space (Gss)	
O	6
S	3
sigma0 / σ_0	2.0159
o_{min}	-1
s_{min}	-1
s_{max}	3
octave / o	{1x6 cell}

Table 1: Gaussian scale space results for image used in figure 13.

Consequently, the algorithm proceeds with the calculation of the difference of the Gaussian (DoG) scale space, which is computed in a separate function as long as Gaussian function execution is completed. The DoG scale space is simply the derivative of GSS scale $G(x,\sigma)$. The DoG represents the same image *I(x)* but at different levels of scale. The DoG scale space is described as follows, and it is designed based on equations 3 and 4:

$$D(x,\sigma(s,o)) = G(x,\sigma(s+1,o)) - G(x,\sigma(s,o)) \tag{28}$$

DoG Scale Space	
O	6
S	3
sigma0 / σ_0	2.0159
o_{min}	-1
s_{min}	-1
s_{max}	3
octave / o	{1x6 cell}

Table 2: DoG results for the image used in figure 13.

The scale σ domain is categorised in a series of logarithmic steps formed in O octaves. Each octave is then subdivided in sub-levels denoted as S. The discrimination between O and S is very important given that each successive octave which is found the data in that octave is spatially down sample by half [1, 7]. Both of them are identified by a discrete octave and sub-level index denoted as o and s, respectively. The octave o and sub-level s are mapped in the corresponding scale σ as follows:

$$\sigma(o,s) = \sigma_0 2^{o+s/S} \tag{29}$$

$$o \in o_{\min} + [0,...,O-1], \qquad s \in [0,...,S-1],$$

,where σ_0 is the base scale level, o is the octave index and s the scale resolution. The existence of octaves with negative index is possible.

For the octave *o* computation the input image is doubled by bilinear interpolation, since the image already been pre-smoothed and $\sigma_n=1$. The DoG scale space is $s \in [s_{\min}, s_{\max}] = [-1, S]$, in order to detect all scales extrema. Also, in view of the fact that the DoG scale space is computed by Gaussian scale space differentiation, the most recent scale's extrema is $s \in [s_{\min}, s_{\max}] = [-1, S+1]$. Thus, the number of octave *O* is set to cover all of the octaves' requirements.

3.2.3 SIFT Detector and Descriptor

As explained in the previous section, the SIFT detector is responsible for frames (keypoints) detection, which has been described in the theoretical background section 2.2.2.3 [1]. The extracted frames are simply a set of image points *(x, σ)* generated from DoG scale space *D(x, σ)* local extremum. Each frame is further characterised by an orientation *θ* which is derived from Gaussian scale space *G(x, σ)*.

The difference between SIFT detector and descriptor is the difference in the "interest" that each one shows in relation to the feature. In SIFT detector, each keypoint *(x, σ)* is weighted by gradient magnitude and a Gaussian window H(x) of deviation $\sigma_w = 1.5\sigma$, while in practice it is cropped at $|x| \le 4\sigma_w$ [1].

The keypoints detection is implemented in a different function, as described in the SIFT DFD illustration in *figure 15*. Keypoints are points of the local extremum of DoG scale space *D(x, σ),* and are extracted in this function by looking at 9x9x9 sample neighborhoods. The element whose value is greater from that of all the neighbors is returned. The return value is not the element value, but the element index. The number of keypoints extracted from each octave when an example image is processed *(figure 13)*, is illustrated as a graph in *figure 16*.

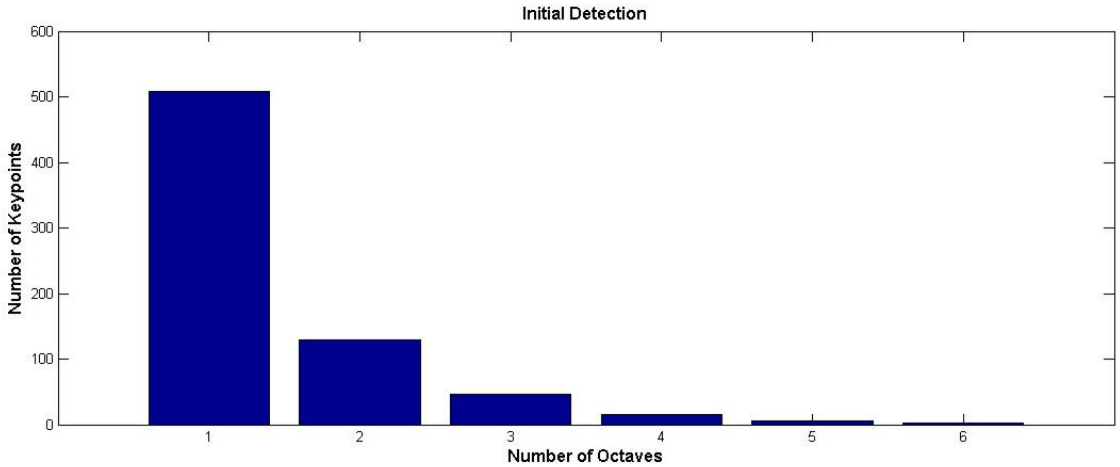

Figure 16: The local maxima keypoint of each DoG octave. The X-axis represents the number of processed octave and Y-axis the number of keypoints extracted. These results comprise the output of function 1 from DFD of figure 15.

Provided that the initial keypoints detection operation is completed, software delete all the points that are close to the binary one, as described in the SIFT function data flowchart diagram step two (2) [7]. The remaining keypoints are processed to the next step (DFD step 3). The number of remaining points can be observed in the graph illustrated in *figure 17*.

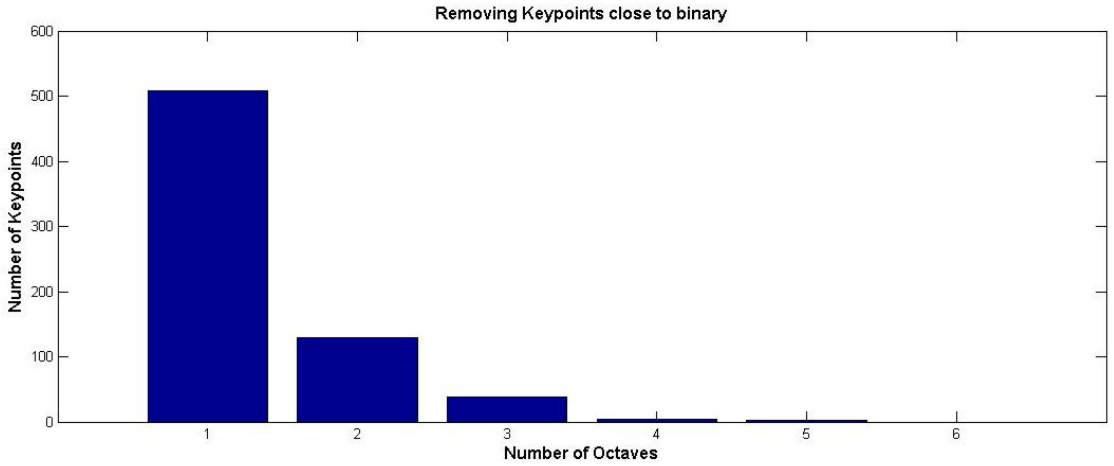

Figure 17: Any points found to be too close to the boundary are discarded. . This process is executed for each octave, but this does not necessarily mean that the results will change every time.

The next step involves the keypoints' refining as well as the weak points' deletion. This procedure is executed in a separate function (DFD step 3). Since keypoints have being extracted from the second step, they are now fitted to the local extremum through a quadratic interpolation process. Concurrently, the weak points are discarded. The process of selecting the weak points is carried out by checking the keypoints' strength, based on threshold *t* value and by examining the histogram maximum of the local extremum [1, 7]. The remaining keypoints number can be observed in the graph below, *figure 18*.

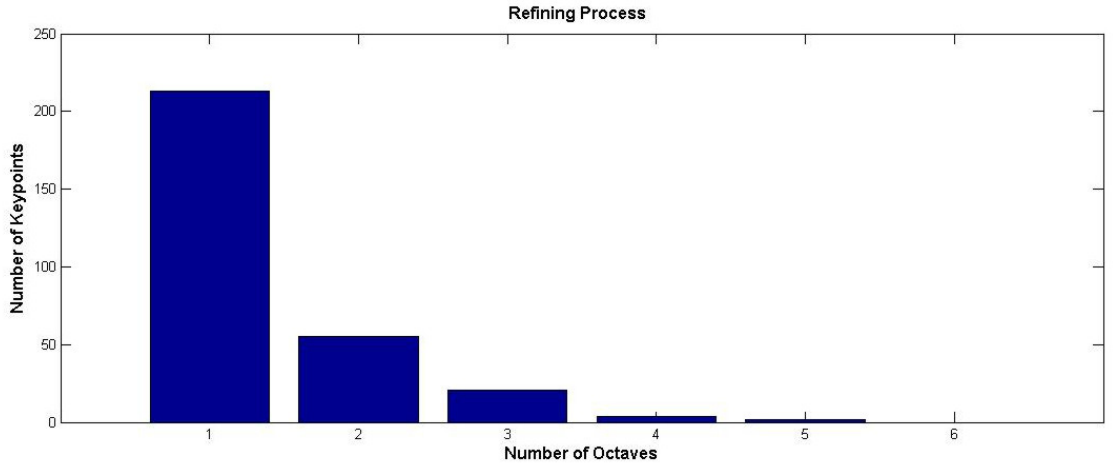

Figure 18: By refining the keypoint location, weak points are discarded by checking the threshold strength and removing points found on the edges.

The last step of the SIFT features detector is the keypoints orientation and storing in the FRAMES matrix. This was introduced in 3.2.1 (Output parameters) section and

illustrated in step 4 of the SIFT function DFD. For each keypoint *(x, σ,)* an orientation θ is computed as the gradient's predominant orientation within a window around the keypoint. This orientation value is computed by finding the histogram of the gradient's orientations maximum within the window around the keypoint. According to D.G. Lowe's implementation, as explained in the theoretical background 2.2.2.3 section, the histogram is weighted the gradient magnitude and the Gaussian window both aligned in the center of the keypoint with σ =1.5 time the keypoint scale σ [1, 7]. The histogram is also pre-smoothed with an average filter. As long as the data are computed in the bins before the histogram's maximum computation, the histogram is pre-smoothed using an average filter. Each local maximum above zero point eight per cent (0.8%) of the global maximum is stored. Thus, it is possible to detect each image location and scale multiple SIFT frames.

As soon as the SIFT detector operation is completed and the keypoints have been extracted and stored in a 4xN matrix, the SIFT descriptors calculation is the next and last step of the function (DFD step 5). Each extracted keypoint is reflected by a descriptor. The SIFT keypoint *(x, σ)* descriptor is defined by the Gaussian scale space octave and it statistically describes the Gaussian scale space $G(x,\sigma)$ gradient orientations. It is a gradient orientation histogram, where the histogram domains are the tuples (*x, θ*). All descriptors are stored in a two dimensional (2-D) array, where each column describes the corresponding extracted feature. As in D.G. Lowe's implementation, each histogram consisted of eight (8) bins in relation to orientation. Thus, N_o = 8 where each descriptor consisted of an array of four (4) histograms around the corresponding keypoint, hence N_p = 4. This leads to the computation of the SIFT feature descriptor vector, whereby $N_p^2 N_o = 128$ [1, 7]. Indeed, as mentioned earlier, this can change since the number of bins N_o can be modified by the programmer. The descriptor vector is returned by the SIFT function, even though it is not always necessary to be used for matching. In general, it is constructive in the evaluation step, when the proposed method is compared with other approaches which also employ SIFT algorithm for image feature recognition.

The resulting extracted features are illustrated in the input image as arrows with one direction.The arrows' length demonstrates feature scale whilst the arrows' direction reflects feature orientation. The results concerning the sampled image are described and illustrated in *figures 19 and 20,* respectively.

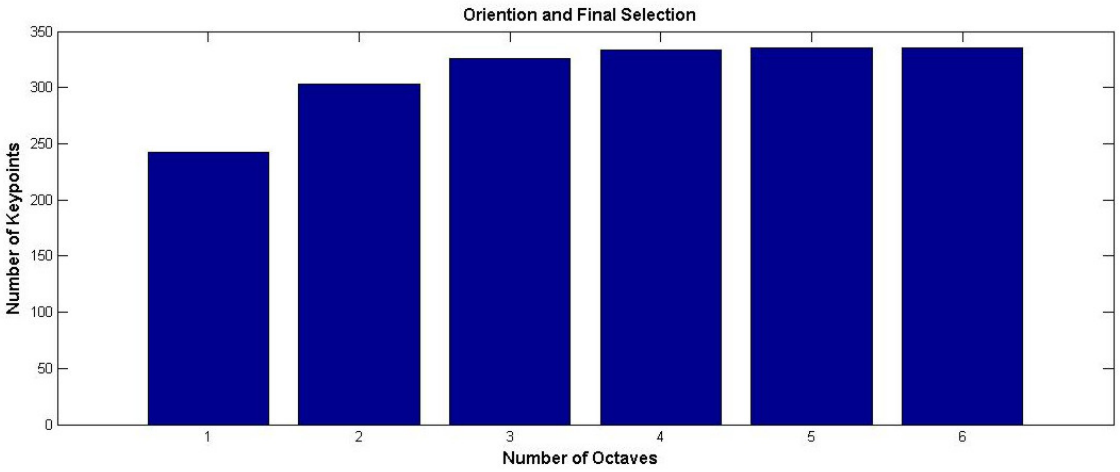

Figure 19: SIFT function extracted image features as computed for each octave, including feature scale σ and orientation θ.

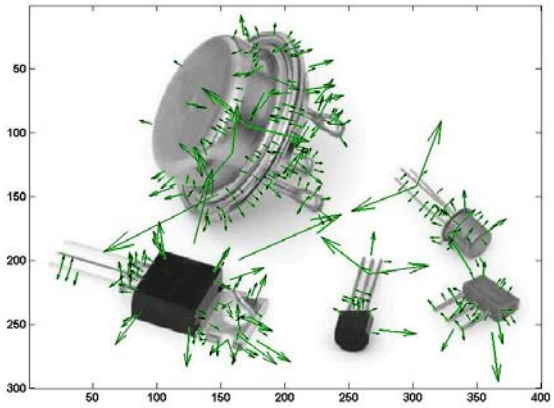

Figure 20: SIFT features as detected and extracted for the sample image of figure 13. 335 keypoints are extracted from input image in 0.002 seconds. These are the results obtained in a complete SIFT execution, involving all the processing steps of SIFT data flowchart diagram.

What is meant by "*Perfect*" image?

In the introduction of this section, mention was made to the need for a *"perfect"* image to achieve the best feature detection. This definition does not refer exclusively to a large and a high contrast image, but to an image with the correct illumination at the correct viewpoint. A new question arises from this one, that is how the user or the company's client know which is the best capture viewpoint and thus to help SIFT to extract the greatest number of possible features. The "*perfect*" image does not refer to the image that the client must upload, but rather to the database image. The database must be updated with images that extract all the required keypoints from the image. An example is an electrical connector with pins and input holes. These connector details must be detected by the SIFT. Nonetheless, if the illumination is insufficient in relation to the detailed points of the components or there is a reflection, then no effective features will extracted, thus leading to bad matching results, as shown in *figures 21 and 22*.

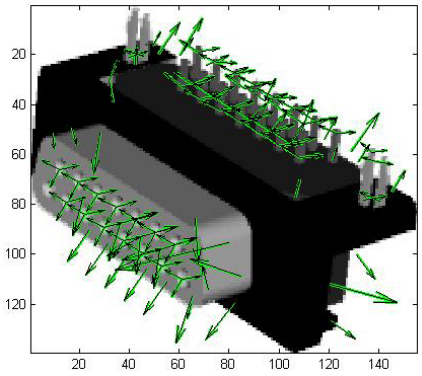

Figure 21: 148 SIFT features are extracted. It is important to observe the symmetry of the features within the component holes and pins.

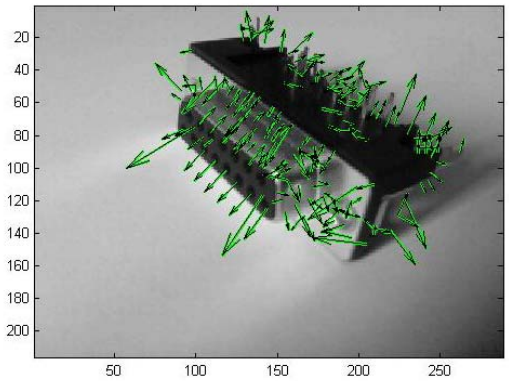

Figure 22: 192 SIFT features are extracted. Only some of the detailed points are extracted and few of them are symmetrical, especially in the pins where reflection exists.

3.3 Match by Correlation (MbC)

This section describes the implementation of match by means of a correlation technique. This technique is also implemented in a separate function. Match by correlation technique is mainly processed when the SIFT operation is over and the distinctive image features have been extracted. This is responsible for the putative (theoretical) matches between the two images' keypoints, as already referred to in the theoretical background section 2.3. During this step, the initial matches between the two images keypoints data set are processed, followed by a second step of point matching, which is explored in the next section, this being the RANSAC algorithm. The keypoint matching in this part of the software is not expected to be one hundred per cent (100%) accurate, since there is no involvement of a distance measurement criterion in the matching selection process. A detailed description of the match by correlation procedure can be observed in the data flow chart diagram of *figure 23*.

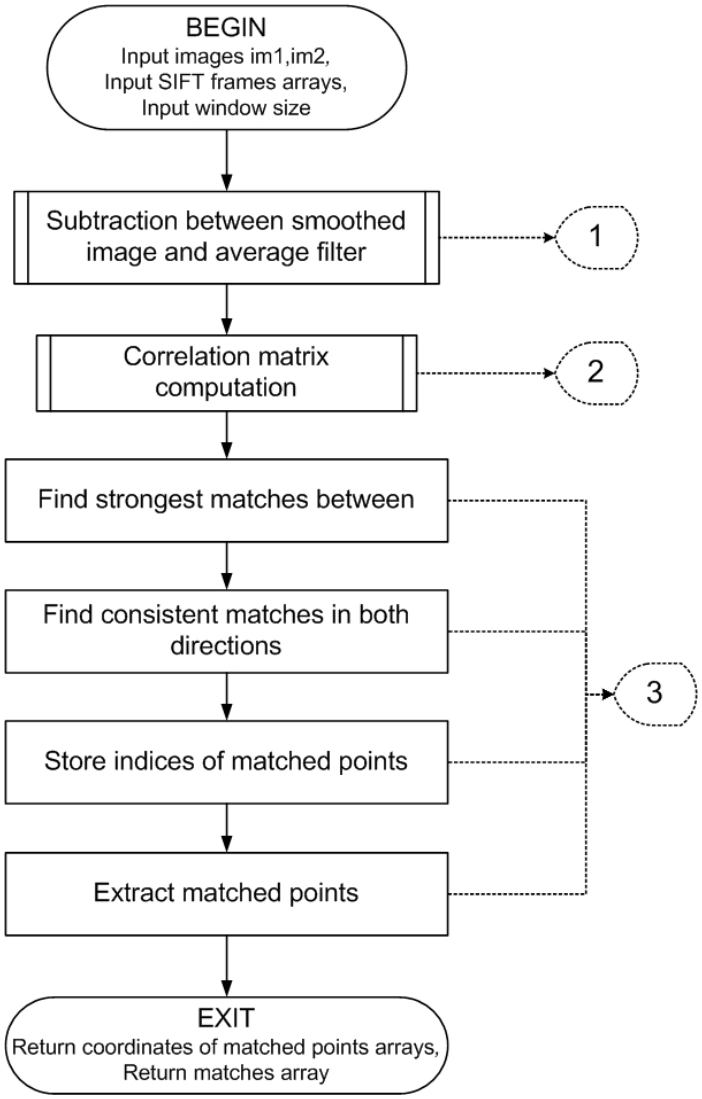

Figure 23: DFD of match by correlation technique function.

3.3.1 MbC Function Input and Output Parameters

Match by correlation function (MbC) is a straightforward function with simple tasks and it involves a short list of input parameters and returned values, as described below:

Input Parameters:

1. The two input images where the SIFT features belong to.

2. The coordinates of the extracted SIFT features from both input images. These coordinates are 2xN number of inputs arrays (f1 and f2), where the first row describes x-axis and the second row describes the y-axis coordinate. The tow arrays of features are not expected to have the same number of points, since the number of extracted features can vary for each image.

3. The correlation window size denoted as *w*. The window size is a square with the center of the window the feature under correlation. Thus, the window value should be an odd number.

Returned Values:

1. The coordinate of points that are considered as putative matches and selected from SIFT feature arrays (f1 and f2). The form of each new array p1 and p2 is the same as that in the input features arrays, where the first row describes the x-axis and the second row describes the y-axis coordinate.

2. The matched points array. This array holds the indices of f1 and f2 array considered to form a putative match. The correlation map is a 2xN array, where the first row gives the number of element points relative to the corresponding feature array for the first image. In a similar way, the second row relates to the second image. This array is solely used at the end of the program (after RANSAC is completed), when the matched point will have been spotted at each image, thus illustrating the matches.

3.3.2 MbC Operation and Correlation Map

The MbC algorithm function processing steps are executed in the DFD sequence shown in *figure 25* and are a replication of the technique introduced above in the theoretical background section 2.3. A second image will be used in this section, with the objective of facilitating an investigation into the working of the MbC technique. This image includes some of the objects of the first image used in the SIFT section, but with some minor changes in scaling and orientation, *see figures 24, 25*.

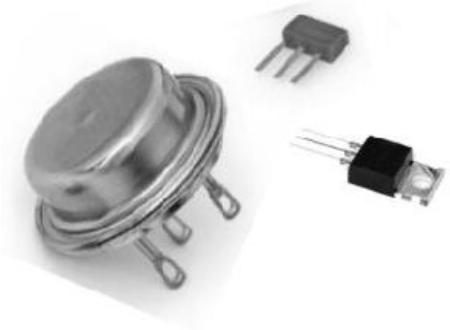

Figure 24: Input image with size of 391x251 pixels. This is the second image used.

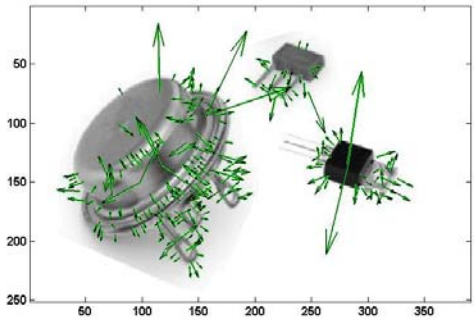

Figure 25: SIFT features as detected and extracted. 245 keypoints are extracted from input image in 0.001 seconds.

3.3.2.1 Average Filter – Step 1

In the first step of the function (DFD step 1), an average filter of w^2 is applied in the pre-smoothed image and subtracted by itself. This filter is a FIR filter and it helps to improve

image contrast by balancing the brightness of each image, thus increasing the efficiency in the correlation process which follows in the next step, *see figures 26(a, b)*.

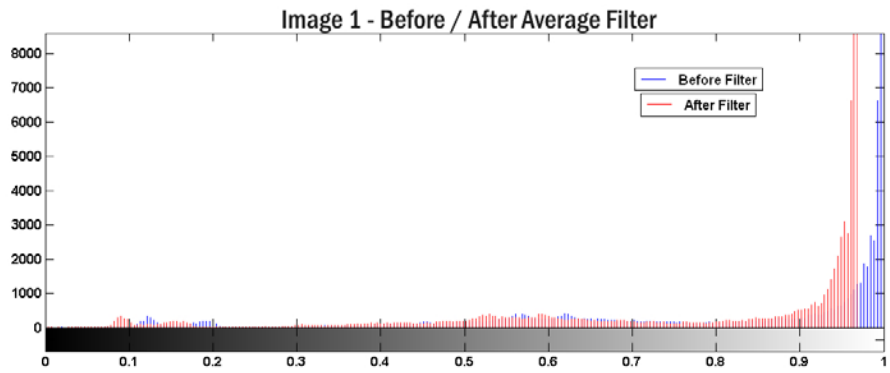

Figure 26(a): Image 1 before and after the FIR average filter is applied. The result shows that the brightness is reduced in order to enhance the contrast.

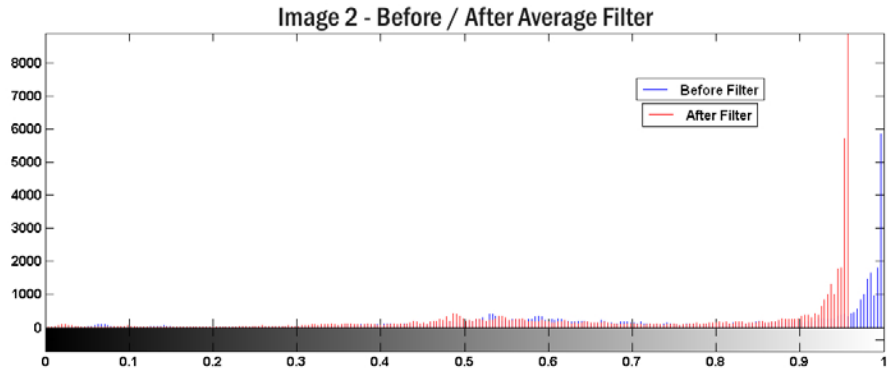

Figure 26(b): Image 2 before and after the FIR average filter is applied. The result shows that the brightness is reduced in order to enhance the contrast.

3.3.2.2 Computing Correlation Map – Step 2

Afterwards, the two images including the corresponding SIFT features matrices, are used as an input for the function that will generate the correlation map (DFD step 2). This function is the most time-consuming part in the MbC function. All the point's correlation computations are performed, as already described in section 2.3.

The correlation map is a function that will compute a correlation matrix. This correlation matrix stores the correlation strength of every image's previously extracted keypoint, relative to every other point of the second image. This computation may increase the application overheat; nevertheless it is indispensable.

At the beginning of the correlation map computation, the correlation matrix values are initialised to infinity. This is undertaken in order to facilitate the later step, during which the computation ends and the non-paired values (those that are still points to infinity), will be ignored.

The next step involves the calculation of correlation windows "radius", *r*. As mentioned before, the window size value is declared by the programmer since it is set as an input parameter which must be an odd number. The computation of this radius is defined as follows:

$$w = (w * 2) - 1, \qquad r = (w - 1) / 2 \tag{30}$$

Provided that the *w* and *r* have been calculated, the processing flow continues with the next step, which consists of finding the point's indices which are at distance *r* from the image boundary points. The point's indices are stored in a 1xN array for each image separately whereby each array is not expected to have the same size.

The next step is the most important one in relation to in correlation windows computation and matching. This step is described in detail in the correlation map function data flowchart diagram, as illustrated in *figure 27*.

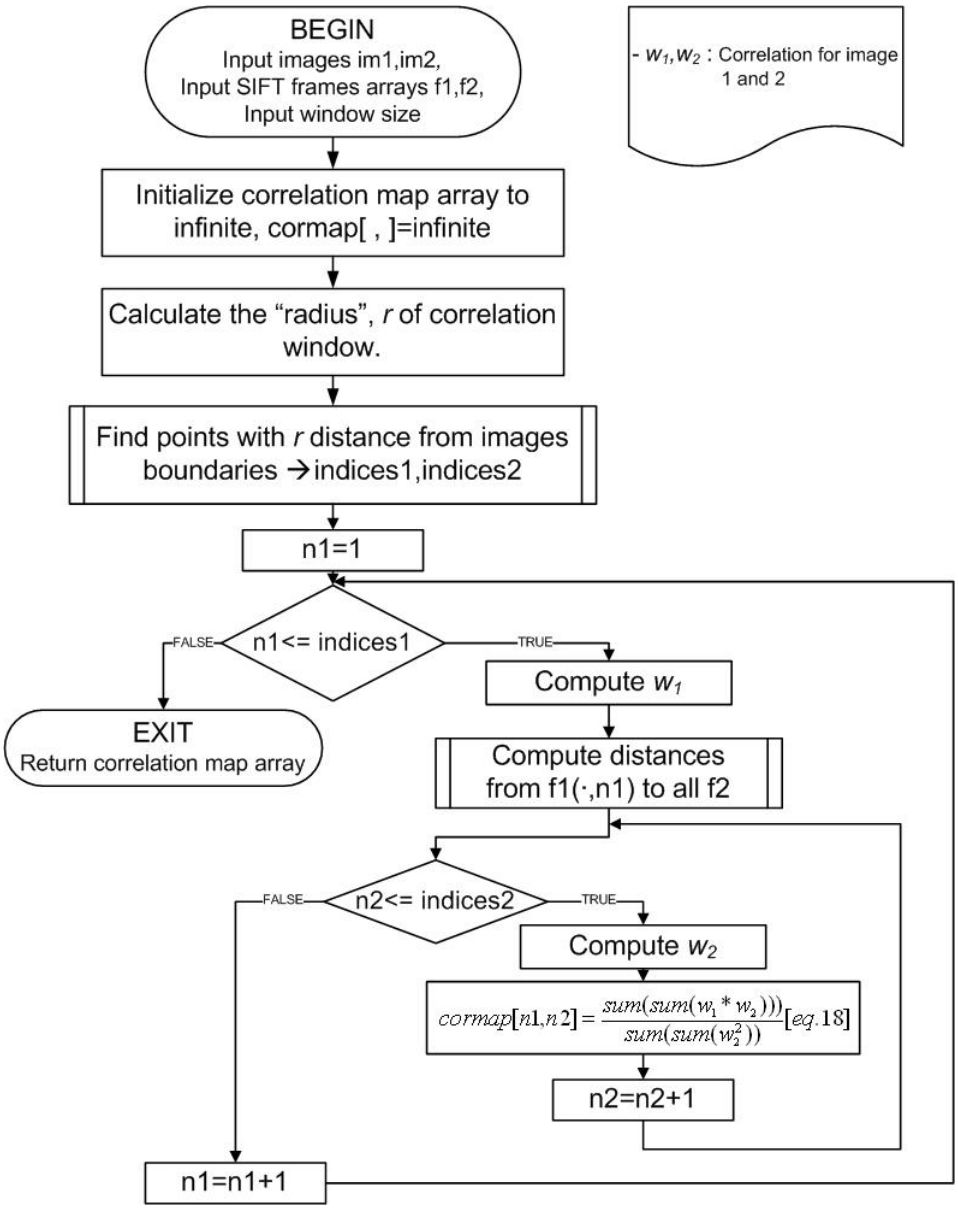

Figure 27: DFD of correlation map generation function. This function is an implementation of correlation technique, firstly introduced in section 2.3.

Once the point's indices have been computed and stored, the software will commence the building of the correlation map array. This step is very simple;yet it is still an essential one given that all of the effective job is executed here. For each point in the indices array a square window of points is generated and pre-normalised to unit vector. This window transformation increases the speed and the accuracy of the result [R16]. Thereafter, the computed point's window is used for the normalised correlation computation in relation to every element from the second image indices array. Normalised correlation is selected since it overcomes the problems faced by the normalised correlation position limitation [R16], as mentioned in the theoretical background in section 2.3.3.

As soon as all the iterations are completed, the correlation map is returned to the MbC function to continue with the points' matching.

3.3.2.3 Finding Strongest and Consistent Matches – Step 3

In this step, where the correlation map is computed, the software will compute the indices of the maximum values along rows in f2 for each f1, in the manner these arrays were introduced in 3.3.1 *Input Parameters* section. The maximum values along columns in f1 for each f2 are also computed. The remaining indices are stored in separate arrays, these being indices 1 and indices 2.

Afterwards, matches which are consisted in both directions will be determined, by checking whether an element of indices 2 array exists in indices 1. If the result is positive, the point is selected and added in indices point array p1 and p2, introduced according to 3.3.1 *Return Values* section. The results obtained in this step are showed below in *figure 28.*

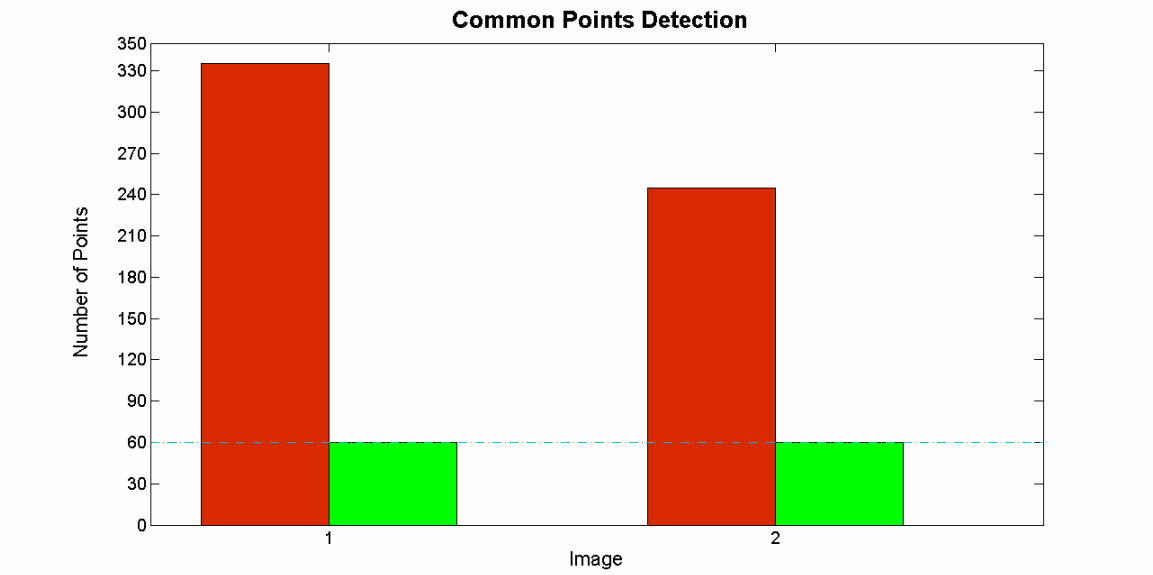

Figure 28: The number of points detected before the matching process is illustrated by bars in red colour for each image. 335 and 245 points have been detected for the first and second images, respectively. After the matching step, it was found that only 60 of them matched (green colour bars).

Once the matching process is completed, the MbC function stores and returns the matched points. Then, the putative matches array is used by the main program to illustrate the common points between the two images, *see figure 29*.

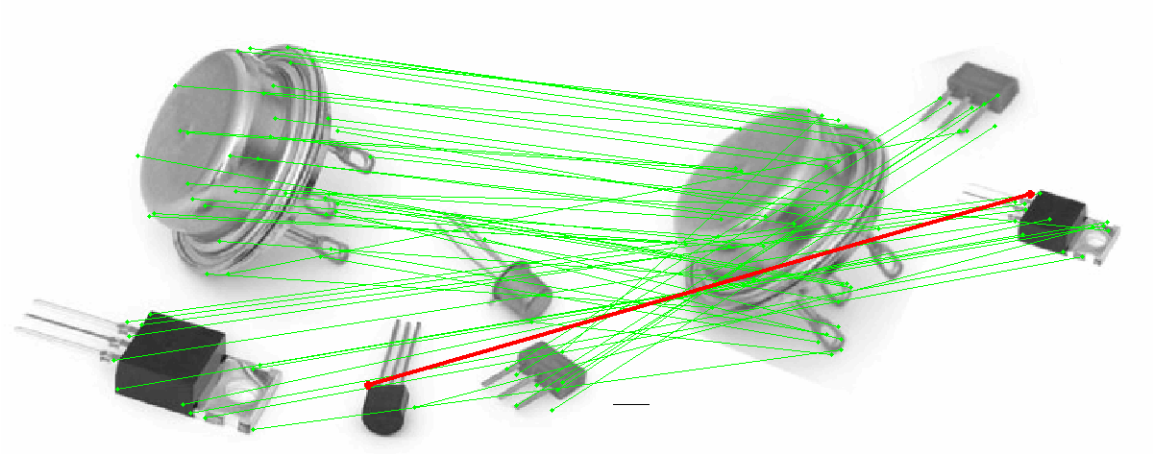

Figure 29: The result of Match by Correlation (MbC) function is illustrated. The results are fairly good (60 points matched), but as it can be observed, there are points that were wrongly detected as common (red line). In this step of the application, a small amount of false matches is to be expected. The whole MbC procedure is completed in 2.411 seconds.

3.2.3.4 MbC Weakness Issue and Solution

As it has already been explained in the theoretical background section, normalised correlation was used instead of un-normalised one, in order to overcome scaling and rotation issues. After some basic tests with the MbC function, it was found that normalised correlation is not invariant. Instead, it is very weak when there is a difference of over twenty degrees (20°) between the images' orientation, and moreover it becomes unstable over fifteen degrees (15°). The conclusion for this weakness arises after some basic test with the MbC function, using images with scaling and rotation difference. The rotation issue can be verified by observing the results obtained from images in figures *30(a-m)*.

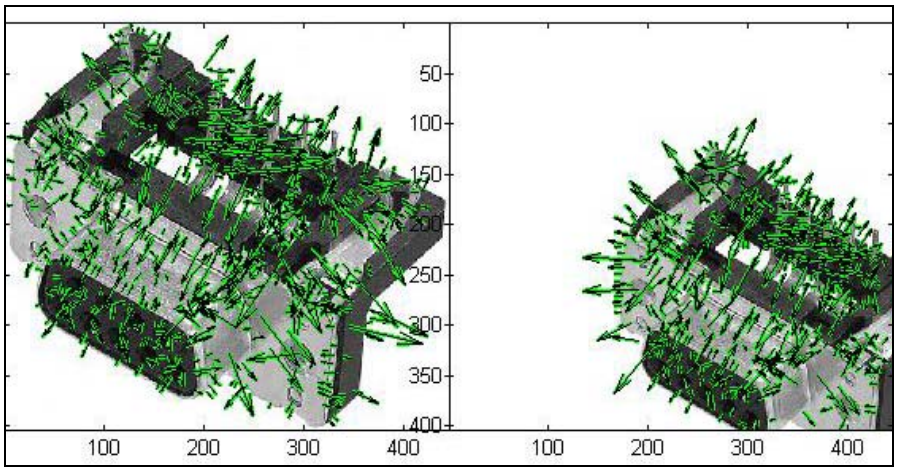

Figure 30(a): The image on the right was scaled and its size was reduced. The above image shows the detection of SIFT frames. Scaling does not affect SIFT frames detection. The number of extracted descriptors for the first image (left side) is 804 and for the second one 508.

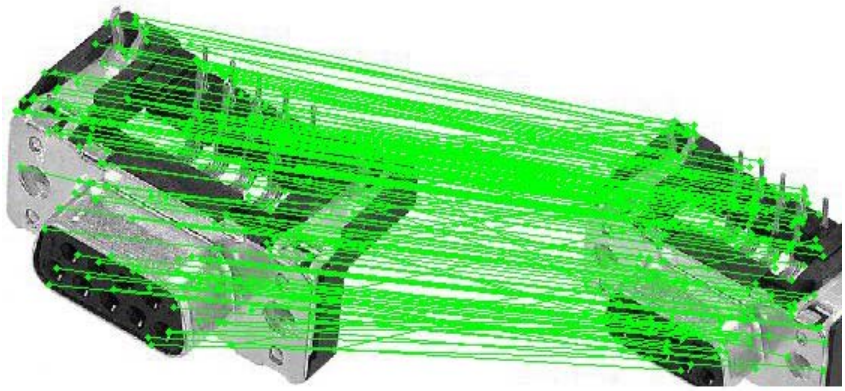

Figure 30(b): Scaling does not affect the MbC technique. The number of putative matches (PM) is 175.

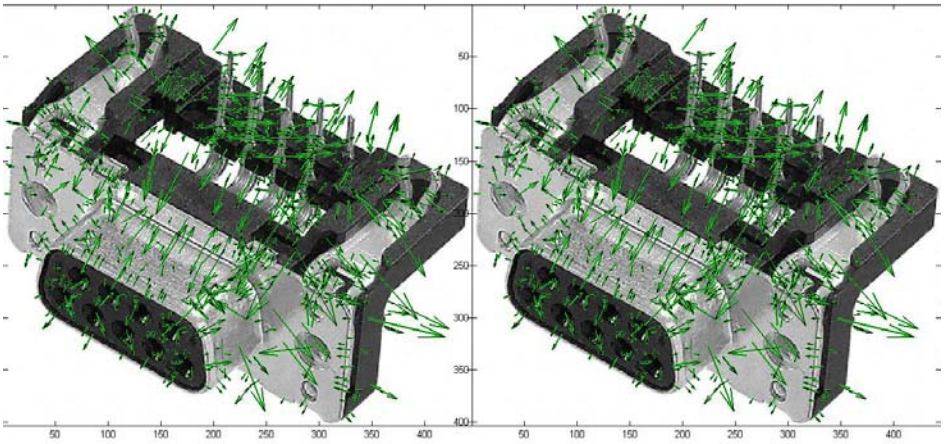

Figure 30(c): In the second test, two identical images were used to examine the rotation issue. The reason behind the use of identical images is to minimise the possibility of matching being affected by other factors. The number of extracted SIFT frames is 804 for both images.

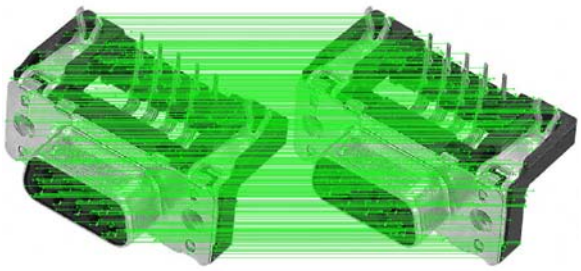

Figure 30(d): Rotation in 0°. **PM:** 706 - **MR:** 88%

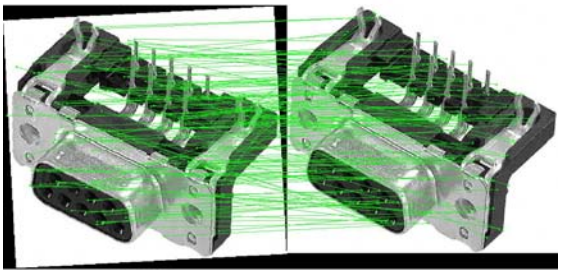

Figure 30(e): Rotation in 3°. **PM:** 141 - **MR:** 18%

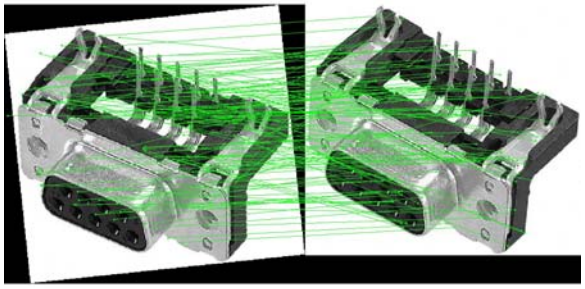

Figure 30(f): Rotation in 6°. **PM:** 121 - **MR:** 15%

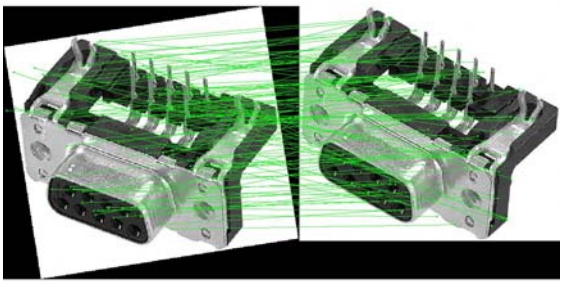

Figure 30(g): Rotation in 9°. **PM:** 107 - **MR:** 13%

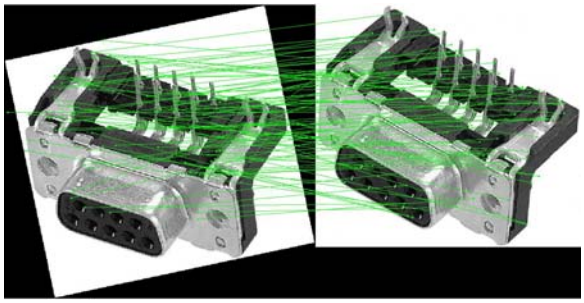

Figure 30(h): Rotation in 12°. **PM:** 86 - **MR:** 11%

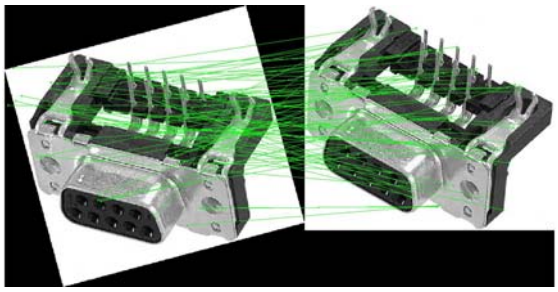

Figure 30(i): Rotation in 15°. **PM:** 78 - **MR:** 10%

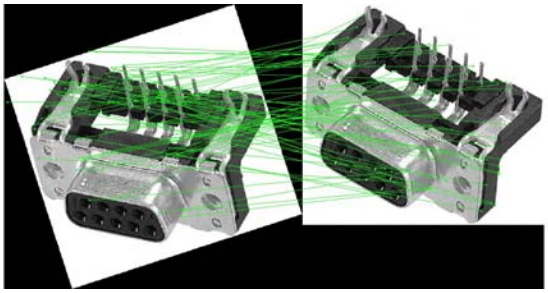

Figure 30(j): Rotation in 18°. **PM:** 65 - **MR:** 8%

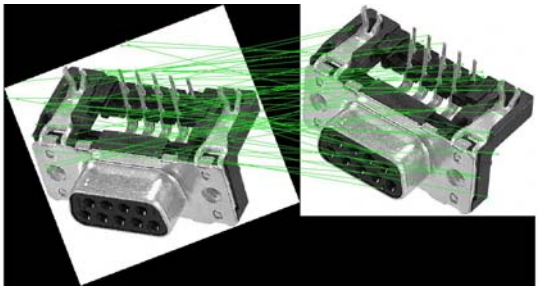

Figure 30(k): Rotation in 21°. **PM:** 52 - **MR:** 6%

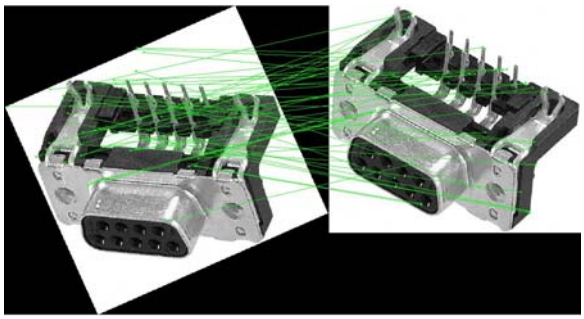

Figure 30(i): Rotation in 24°. **PM:** 47 - **MR:** 6%

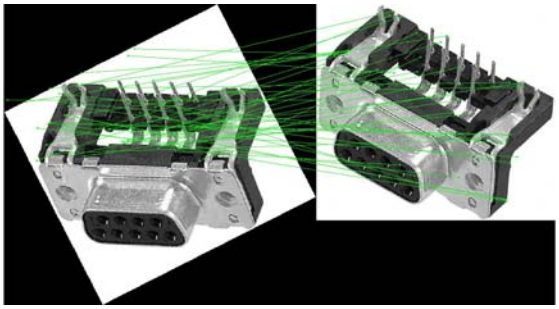

Figure 30(m): Rotation in 27°. **PM:** 43 - **MR:** 5%

PM = Putative matches number
MR = Matching Rate

About this example: *The matching rate is calculated by using a percentage formula. Since the number of extracted points is the same in both images, the matching rate is calculated with the following formula:*

$$MR = \frac{100 \times PutativeNumber}{FramesNumber} \tag{31}$$

In this example it is important to observe the matched point's dispersion. The green line defines a matching between two points. As rotation increases in the first image, the matching ratio is reduced and the accuracy starts to fail.

The solution of this problem is achieved by rotating the window before it is correlated with the second image window. Each rotated window will form a different correlation map, and in the end the "strongest" map will be returned for the matching procedure. The new question which arises is what is the criterion to be used by the software in order to

decide which is the "strongest" correlation map? In each rotated window (image 1), the software will check the extent of correlation in every window within the second image. In the end, the degree value with the most successful correlations will be selected. This check will be performed by using one of the MATLAB build-in functions for the computation of a two dimensional (2D) cross correlation coefficient [28]. This function is simply the programmatical implementation of equation 18. The efficiency and generally the evaluation of this method will be discussed and explored in the last sections of the evaluation and conclusions discussion.

As previously explained, each window will be rotated before being correlated. The significant issue which arises from this rotation is that the window size will change an outcome which will lead to the forced and unsuccessful end of the application. For this reason, each rotated window is also cropped and thus keeps its original window dimensions, as calculated in the beginning. The way that the windows will look like is showed in *figures 31 and 32*.

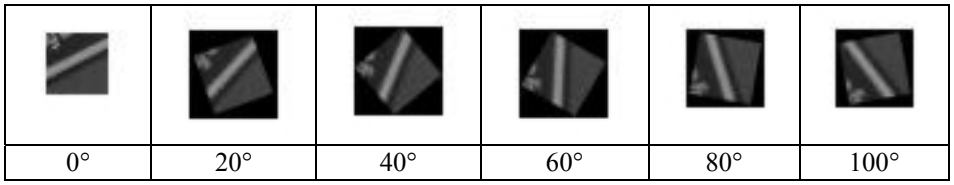

Figure 31: Windows of the first image after "pure" rotation. The first window from the left (0°) is the original window before rotation.

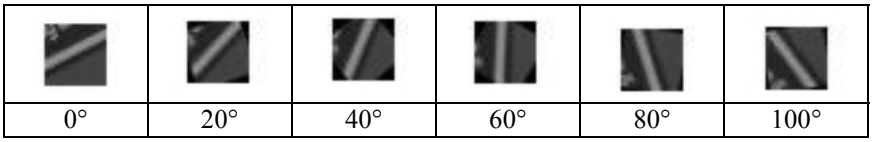

Figure 32: Rotated windows of the first image are now cropped. There is no change in the first window from the left (0°), since it is formed as a complete square. As the window rotation comes closer to the four quadrants (90°, 180°, 270°, 360°), the existence of black (*dead*) pixels is reduced.

Observing the results, it becomes clear that in view of the fact that the window is cropped, some of its information will be lost. The lost information is replaced by white or black pixels. Since the pixels are white or black, this means that the correlation coefficient computation will produce different results, which affect the correct angle selection.

To overcome the above problem, a larger window is selected, which is approximately 1.5 times larger that the original one, which is rotated and cropped. Because this window is larger it holds more information, thus, when it is cropped the window which is produced will be formed only by image pixels, without the addition of any "dead" pixels (this being a pixel added to replace the cropped one). The correlation coefficient calculation will be computed between the cropped window and the second image window, both having the same size. In the correlation step, the initial small-cropped window must be used; otherwise the reference to the image keypoints will be wrong. The window

transformation is illustrated in *figure 33* whilst the window matching procedure is represented in *figure 34*.

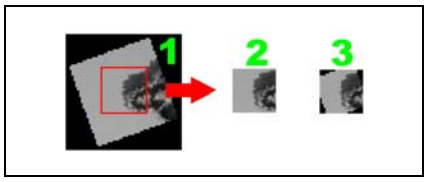

Figure 33: The window on the left side is the window (1) that is used for computing the second window (2), so as to overcome the problem of black pixels, and thus wrong calculations in correlation coefficient. The third window (3) will be employed once the correlation map has been build.

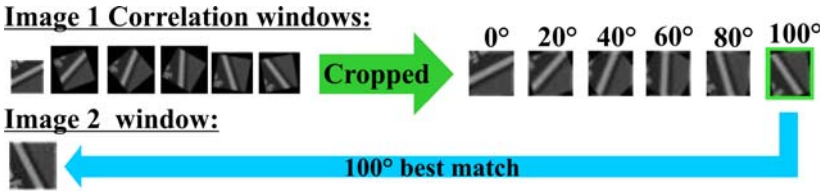

Figure 34: In the above example, during the window matching procedure, the window which has a 100° rotation is selected. The comparison with the window of the second image is performed using correlation coefficient technique, as this is described in equation 18.

As mentioned before, as the rotation in an image comes closer to four quadrants, the number of black pixels added in the rotated correlation window is reduced. Thus, more image details are correlated. On the other hand, as the image rotation comes closer to the middle of a quadrant the number of added pixels increases, and a result of this image details are lost. In this case, the availability of the keypoints is reduced since some of them are lost. On the basis of this fact, the number of matched points is also reduced.

By using the image of the previous example, the new updated MbC technique is tested, examining its behavior in the context of large image rotation, *see figures 35-39(a,b)*. More tests and results will be explored in the course of the Testing and Evaluation section.

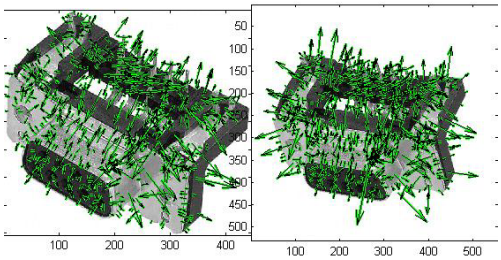

Figure 35(a): The second image (above on the right) is rotated by 20°. After SIFT execution 804 features were extracted for the first image and 925 for the second.

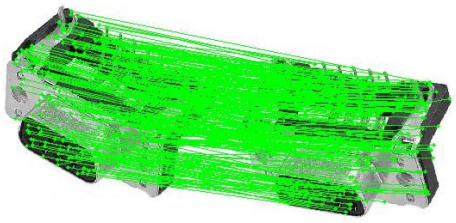

Figure 35(b): After MbC execution 327 points were found common between the two images, and these are spotted in the image above.

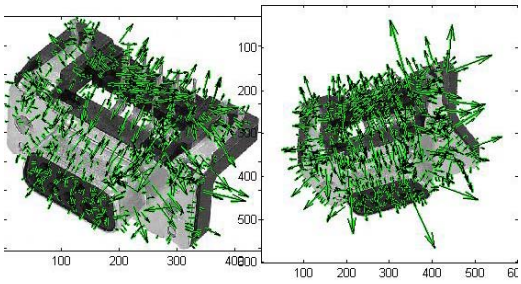

Figure 36(a): The second image (above on the right) is rotated by 45°. After SIFT execution 804 features extracted for the first image and 941 for the second.

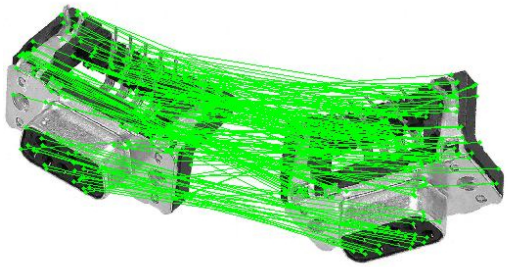

Figure 36(b): After MbC execution 191 points were found common between the two images, as these are spotted in the above image.

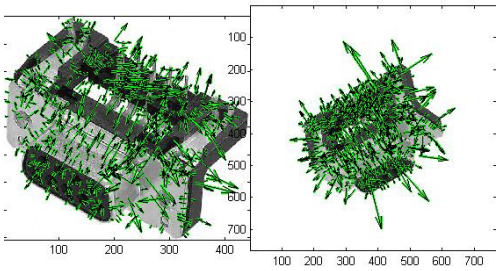

Figure 37(a): The second image (right) is rotated by 60°. After SIFT execution 804 features were extracted for the first image and 925 for the second.

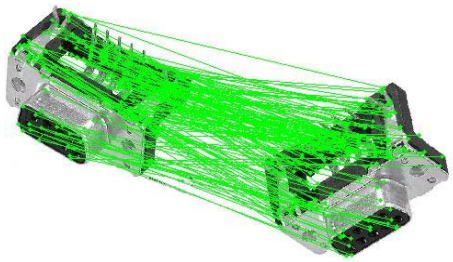

Figure 37(b): After MbC execution 154 points were found common between the two images, and these are spotted in the image above.

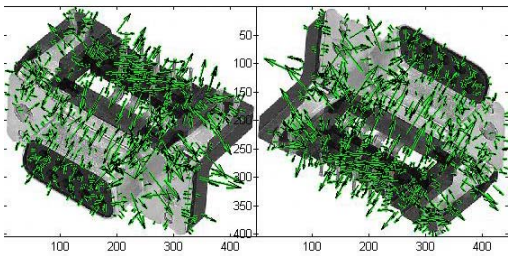

Figure 38(a): The second image (above on the right) is rotated by 180°. After SIFT execution 804 features were extracted for the first image and 872 for the second.

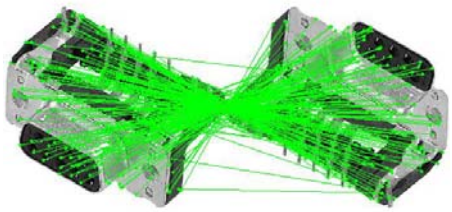

Figure 38(b): After MbC execution 210 points were found common between the two images, and these are spotted in the image above.

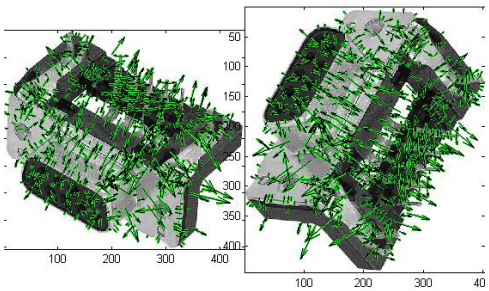

Figure 39(a): The second image (above on the right) is rotated by 270°. After SIFT execution 804 features were extracted for the first image and 869 for the second.

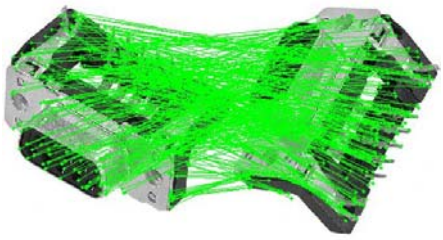

Figure 39(b): After MbC execution 296 points were found common between the two images and these are spotted in the image above.

3.4 RANSAC Features based Matching Verification

This section describes the implementation of the RANSAC algorithm. RANSAC is implemented in a separate function and it is responsible for estimating the inliers from the putative matches' data set. Since the data set comes from putative matches, a considerable percentage of this set will be rejected by RANSAC.

The RANSAC algorithm involves the collaboration between the Epipolar geometry and the Fundamental matrix model. In this collaboration, the Epipolar geometry technique is firstly used to compute epipoles between two images, and then the fundamental matrix model steps in to encapsulate the intrinsic geometry [17, 29]. In other words, the fundamental matrix is the algebraic representation of Epipolar geometry [29]. As soon as the inliers set has been computed, the matching rate between the two images is calculated to define whether the two images share common objects.

This procedure is separated into two parts, these being the general computations and the RANSAC algorithm (probabilistic part) implementation. This is necessitated by the fact that the epipoles as well as the fundamental matrix must be computed first. For this reason, it was deemed as more convenient and efficient to fragment the program into in smaller parts.

The algorithm (probabilistic part) operation is illustrated in the data flowchart diagram (DFD) in *figure 11,* and explained in section 2.4.3 of the theoretical background. Both RANSAC procedure parts are illustrated in *figures 40 and 41*, respectively.

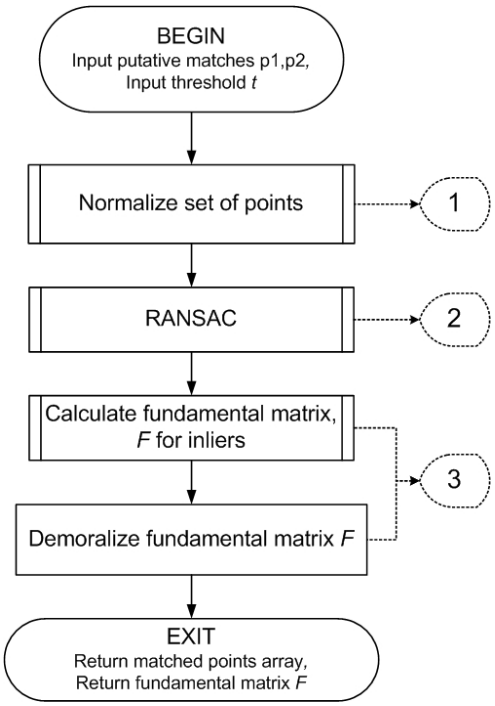

Figure 40: DFD of main RANSAC function, as this is called by the main program.

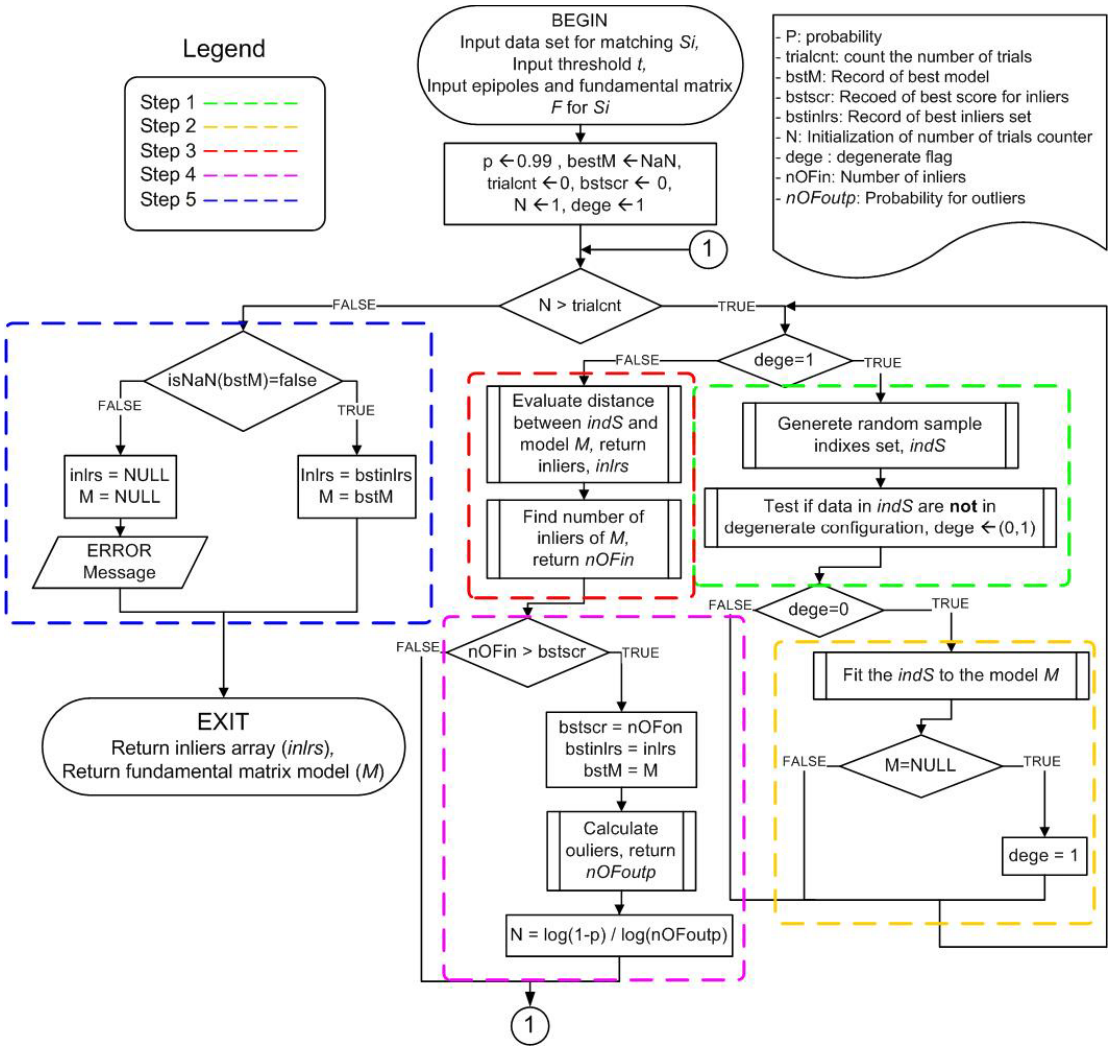

Figure 41: Detailed DFD of RANSAC inliers computation, as this is performed in the second (large) function.

3.4.1 RANSAC Function Input and Output Parameters

Input Parameters:

1. The coordinates of the putative matches points for each image, which are p1 and p2. These arrays are the output of the MbC function, which located the putative matches between the two images.

2. A threshold value, *t*. This is the distance threshold between the putative keypoint and the computed model. This threshold will be used as the criterion to decide whether a point is an inlier or not. The importance of this parameter is that the point coordinates are normalised to the mean distance from the origin. Thus, the threshold value must be set relative to this between 0.001 – 0.01.

Returned Values:

1. A 3x3 fundamental matrix can compute and illustrate the epipoles lines, especially where verification is needed.

2. An array of inliers. This is a one dimensional (1D) array which holds the inliers that come from the putative matches' arrays. Because it is a one dimensional (1D) array, in the end it will be decoded into a two dimensional (2D) array, and hence, it will include the coordinates of each inlier.

3.4.2 RANSAC Function Operation

3.4.2.1 Normalizing the Data

As it can be seen from both data flow chart diagrams, the most complicate part of the algorithm is executed in the second function, where the actual operation of the RANSAC algorithm is processed. However, this operation is simple to understand as long as the Epipolar geometry and fundamental matrix model theories have been comprehended.

The RANSAC function starts with a normalisation procedure, where each set of the putative matches is normalised, so that each set's center is at the origin and their mean distance from the origin is $\sqrt{2}$. This normalisation technique is used to improve the conditioning of any equation involved and to solve fundamental matrices.

3.4.2.2 RANSAC Main Operation

As explained in section 2.4, the RANSAC algorithm is a probabilistic algorithm, which selects a random data set S_i within the putative matches' data set, and estimates the number of inliers. This procedure will be executed until the greatest number of inliers is found relative to the probability value that has been set. The probability value is not given as a function argument, and consequently it is not in the user option. Therefore, it has been set to the best probability so that a data set S_i with the most inliers is selected.

The main operation consists of five (5) steps. Each step is composed of other smaller steps (function). The structure of these steps and their order of steps' execution is described in the DFD of *figure 42* (colored lines).

Provided that the parameters have been initialised, the algorithm is executed. This execution uses two (2) loops. The first loop will be repeated until the probability of the largest set of inliers is found and every computation is involved in this loop. The first step of this iterative process (first loop) is the generation of a random data set, consisting by eight (8) points. The degeneration procedure (DFD steps 1-4) starts by generating a random data set S_i and testing whether these points within the data set are not in degeneration form (step 1). The degeneration test is performed to determine whether the

selected set of the matched points will result in a degeneracy form in the fundamental matrix model calculation, as required by RANSAC [35].

If the set S_i is degenerate, then the randomly selected data will be used in order to build their fundamental matrix model. The resulting model will be checked whether it has a successful fit or not. Moreover, this is a good way to determine if the data is degenerate or not (step 2).

When the degeneration fails a model is formed. The computed model is verified by checking the distances between the points and the indices returned by the model. The length of the model returned indices is computed thus defining the number of inliers (step 3).

In the next step (step 4), the program checks whether the computed inliers number is the best so far. If the result is positive it stores the current fundamental model and corresponding inliers for the next check. The number of trials is now computed with the objective of ensuring that the data set S_i was selected with probability p and does not include any outliers. The calculation of this probability is defined by the following equation:

$$N = \frac{\log(1-p)}{\log(p_{outliers})} \tag{32}$$

Provided that the probability of selecting the most successful model is satisfied and the inliers set has been computed, the next step is to determine whether the given solution is valid. This can be achieved by simply checking the model data in order to decide whether they are valid numbers. If the numbers are valid, the best model together with the best inliers set is returned; otherwise, a null model and a null inliers array are returned, thus notifying that there are no valid matches between the two images (step 5). An example with a large number of outliers, where RANSAC is used for inliers estimation can be observed in *figure 42(a,b)*.

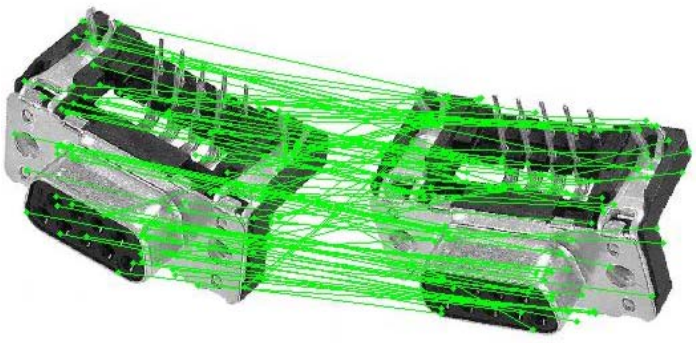

Figure 42(a): In this image, 103 putative matched points are found. The correlation window technique is disabled, thus creating a larger number of outliers which test the RANSAC performance.

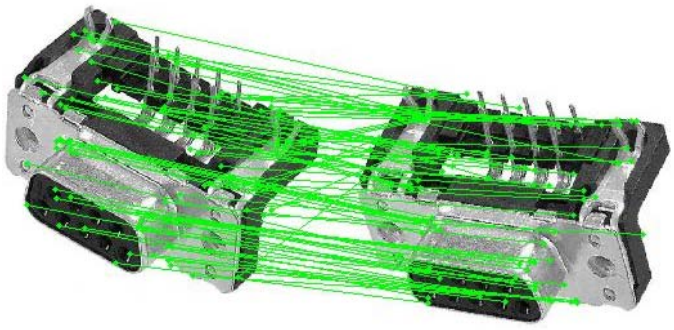

Figure 42(b): In this image, 83 inliers are found and 20 putative points are discarded. Based on the matching rate formula of equation 31 the matching rate is estimated at 81%.

3.4.2.3 Calculating and Demoralizing the Fundamental Model

After the computation of the fundamental model and the inliers of the given set, the fundamental matrix of the inliers set relative to the putative matches is computed. This fundamental matrix computation is essential because it can be used in the matching verification. With this computation, the Epipolar line can be calculated and illustrated between the two images, thus making possible the observation of the matched points one by one, as shown in *figure 44(a, b)*. The fundamental matrix must be demoralised before it is returned to the main program. If the fundamental matrix is denoted with F, then its demoralization is defined by:

$$F = N_2' * F * N_1 \tag{33}$$

where N1 and N2 are the normalised matrices computed relatively to their putative matches sets, this being image 1 and image 2, respectively.

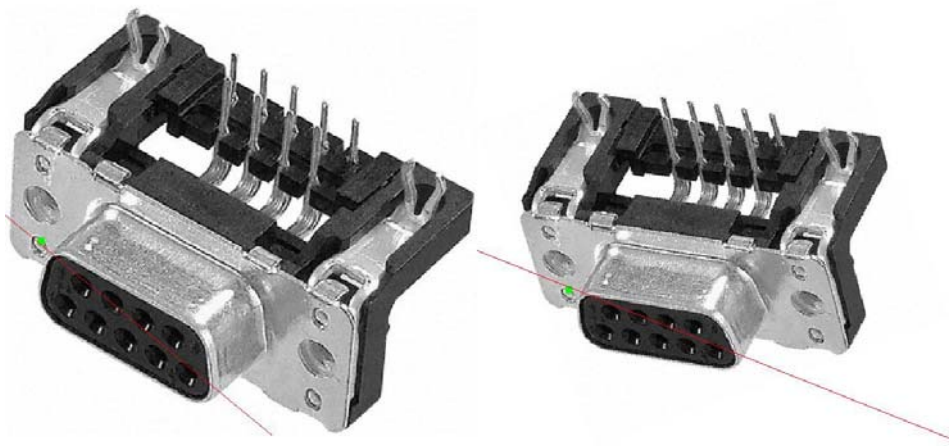

Figure 43(a): Epipolar line 1.

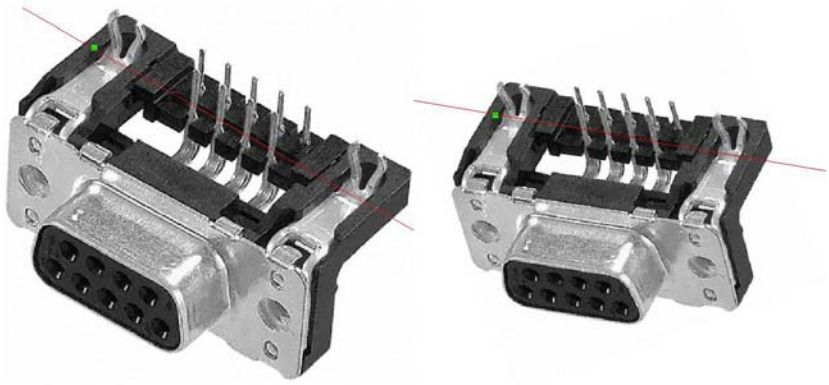

Figure 43(b): Epipolar line 2.

Figure 43(a, b): Epipolar lines for the first and second (20° rotation) images. The calculation of epipoles for both images is performed after the RANSAC completion. The number of Epipolar lines that can be drawn can be as many as the number of inliers found. In this example there are 83 lines.

3.4.2.4 Matching Rate Calculation

After the RANSAC function completion, the putative matches and inliers sets are employed for the calculation of two images matching ratio. This rate value is used to define whether two images have common objects. Even if the RANSAC returns a set of inliers, this does not mean that the two images are common. For this reason, it is important that a matching rate is to be calculated. Consequently, its values are used as a matching threshold to define whether the two images match. Depending on the three user requirements and by setting the matching ratio value as a threshold where values below that are considered as "not matched", the user can make the application as distinctive as he wants.

The matching rate's (MR) calculated approach is the same one as in equation 31. The result is obtained by using the following equation:

$$MR = \frac{100 \times InliersNumber}{PutativeNumber} \tag{34}$$

The matching decision is not based only on the result of this formula. Assuming that an image I1 is compared with images I2 and I3, and the matching ratio of I1-to-I2 is seventy nine per cent (79%), with 480 putative matches and 380 inliers, and the result of I1-to-I3 is ninety per cent (90%) with ten (10) putative matches and nine (9) inliers, the application will recognise and select I1-to-I2 as the most successful matching. The selection of the correct matching decision cannot be based only on the MR result, especially when an image is compared with more than one, like the company example where it will be examined with thousands of images. However, the number of inliers found will be considered to determine the most accurate matching result. More test examples will be explored and discussed in the next section.

4. Experimental Results and Analysis

In this section, the proposed method for object matching is tested and any result obtained is statistically analysed. Every test contains all of the three algorithms; SIFT [1], Cross Correlation Matching (MbC) [25] and RANSAC [9]. The correlation window of MbC method was set to 23x23, a random large size, thus to increase the performance when there is a large translation between the compared images. All tests are implemented in MATLAB R2010a and the proposed method will be tested under different images conditions, as image rotation, translation and large scaling. Images used for any other equivalent object matching technique will be used in this section, thus to help in the evaluation section for the comparison of the proposed method with its predecessors. Also images captured from real electrical components supplied by RS Company are tested. Some of this components contain a defect, thus to test how efficient and accurate the application can be when details from an object have been destroyed. Additional tests are however described through this section.

During all the tests, the computer with the follow requirements was used:

- Intel Pentium M 725 at 1.6 GHz processor
- 1.5 GB RAM of 166 MHz max bandwidth

For each test application, both illustration results and a feedback report is provided, see figure 44. The illustrations are presented in each test's results section and the feedback report is converted to a table, thus only the resulting important information is collected. The feedback report has the follow structure:

```
----------RS Project----------
1.Loading images...
  Images loading completed. Time: 0.176 sec
2.SIFT operation started...
  SIFT operation completed. Time: 0.003 sec
  -> Image 1 # of descriptors: 1034
  -> Image 2 # of descriptors: 375
3.Matching operation started...
  -> Correlation Match algorithm is running...
**** Window Size: 23  → This is the correlation window size
  Matching operation completed. Time: 10.083 sec
4.RANSAC operation started...
  -> Number of inliers: 62 (82%)
  -> Number of theoretical matches: 76
  RANSAC operation completed. Time: 0.095 sec
  ->Result: OBJECTS MATCHED
5.Exit project. Total time: 0.2 minutes
----------END----------
```

Figure 44: Sample of feedback report that application provides while processing.

4.1 Testing Using Random Images

In this section, randomly selected images as buildings, landscapes and images that used for the evaluation of other object matching applications will be testing the effectiveness of the proposed method. *See figures 45(a-c)*.

4.1.1 Identical Images Affected by Scaling and Translation

In this example, two identical images were used. The second image was affected by scaling, translation and change in perspective. *See figures 45(a-c)*. Both images are converted to grayscale when the application starts, as it is explained in 3.1 (step1).The images' properties are shown in table 3:

Properties	Image 1 (Left)	Image 2 (Right)
Format	JPEG	JPEG
Initial Size	640x480	640x480
Color	RGB	RGB

Table 3: Test 1 images properties.

4.1.1.1 Results

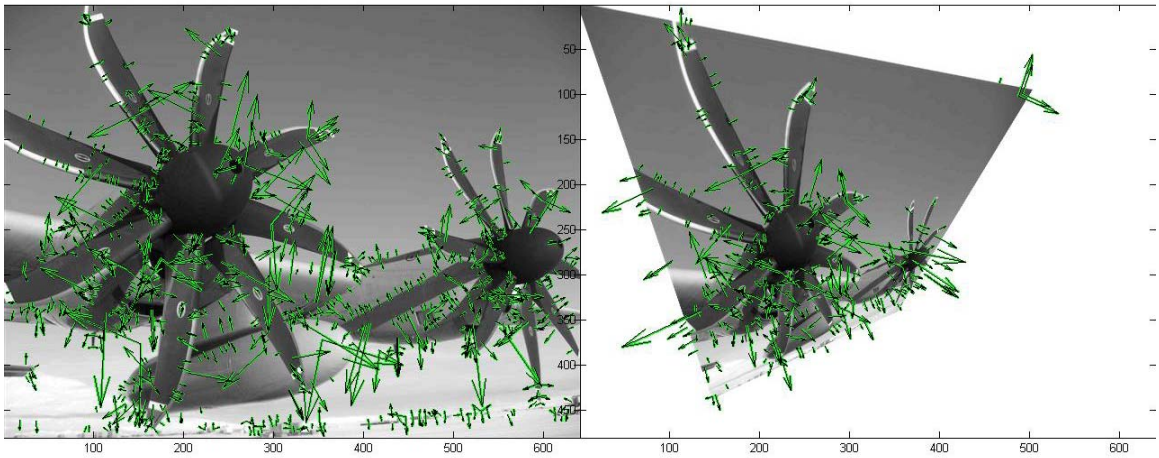

Figure 45(a): Shows the extracted SIFT frames for each image. Because the second image is affected by scaling, translation and change in perspective, the image pixels form has changes, thus SIFT to detect some different frames than in the first original image (Right image at left down and right top corners Each arrow describes a SIFT frame, where the arrow length and orientation is the frame scale and orientation.

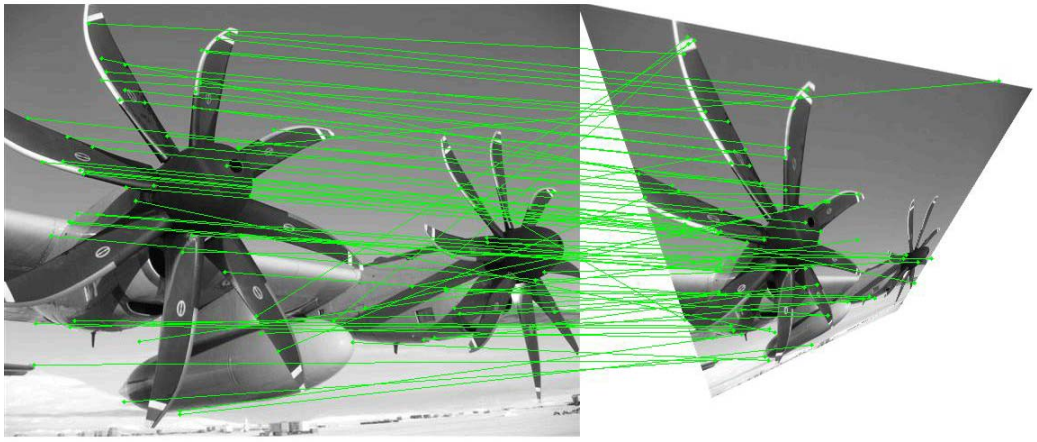

Figure 45(b): Shows the interconnection of points that MbC technique considers as putative matches between the two images. There are some wrong matches between the two images. In this step of the program, this was expected.

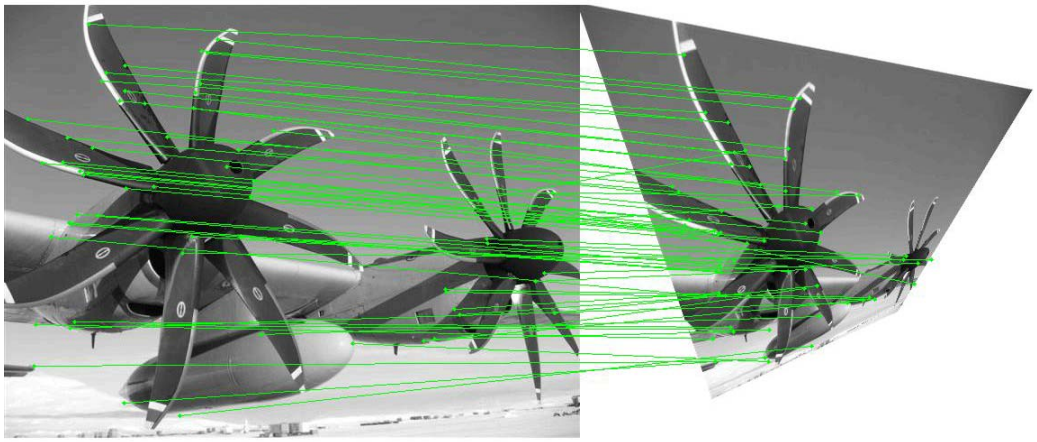

Figure 45(c): Shows the RANSAC algorithm results. The remaining points (from putative matches set) are the estimated inliers.

	Image 1 (Left)	**Image 2 (Right)**
SIFT – Frames detected	1034	375
MbC – Putative matches	76	
RANSAC – Inliers estimation	62	
Matching Rate (MR)	82%	
Status	Images Matched	

Table 4: Shows the SIFT, Match by Correlation and RANSAC results for test 1.

	Execution Time (Seconds)
SIFT – Frames detected	0.003 s
MbC – Putative matches	10.083 s
RANSAC – Inliers estimation	0.095 s
Application Total Time	12.21 s (0.2 minutes)

Table 5: Shows the SIFT, Match by Correlation (MbC) and RANSAC execution time in seconds. The application total time includes any other function call.

4.1.1.2 Analysis

The results obtained (tables 4, 5) from the first test shows that the proposed method can effectively extract and detect the common points between the two images. The SIFT algorithm results show that image scaling, translation and the change in the viewpoint affecting the number of the keypoints that can be extracted. However the matching results show that SIFT is invariant in these three image transitions. In Match by correlation (MbC) technique, the optimisation performed in correlation windows for invariance in rotation was disabled, thus avoiding the adding of extra overhead. Even if the rotation optimisation was disabled, the MbC matching was fairly accurate since only ten (10) of the putative matches were discarded by RANSAC with a matching rate of 82%. Considering time execution, the SIFT and the RANSAC algorithms are very fast with execution time less than a 1/10 of a second. The MbC method is responsible for most of the time overhead since it has the higher rate in execution time of about 200 time more than the average of SIFT and RANSAC execution time. This high overhead rate is because of the fact that MbC method is processing the image iteratively until the putative matches are computed.

4.1.2 Landscape Images Affected by Rotation

In this example, two landscape images of the same plan but of different perspective were used. Both of the images were captured in a shadowy environment, thus to reduce the image brightness. The second image was rotated by a 100°, thus to create a large difference between the orientation of the two images, figures 46(a-c). Both images are converted to grayscale when the application starts, as it is explained in 3.1 (step1).The images properties are shown in table 6:

Properties	Image 1 (Left)	Image 2 (Right)
Format	JPEG	JPEG
Initial Size	730x547	814x666
Color	RGB	RGB

Table 6: Test 2 images properties.

4.1.2.1 Results

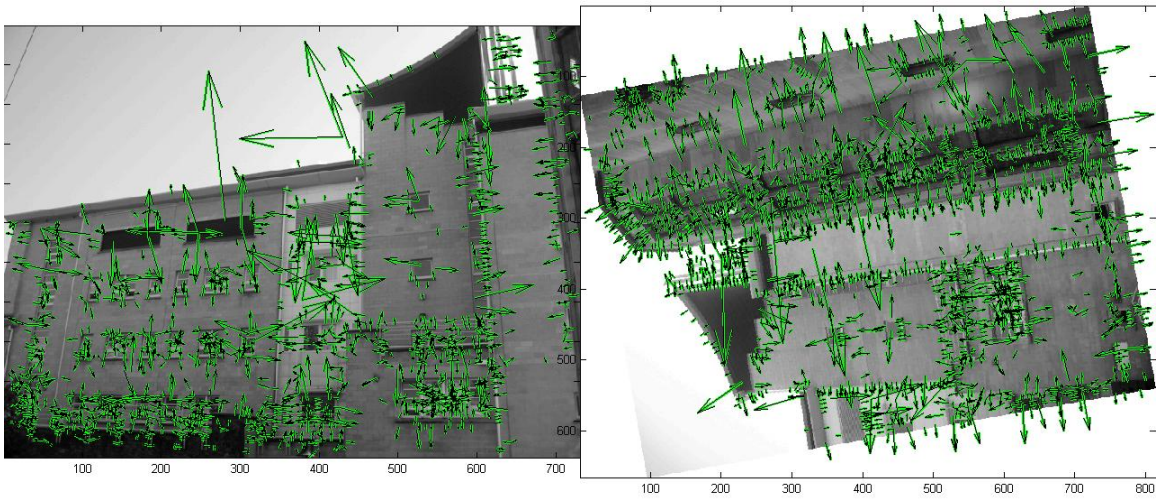

Figure 46(a): Shows the extracted SIFT frames for each image. The second image is affected by 100° rotation. The SIFT shows its invariance in rotation. Each arrow describe a SIFT frame, where the arrow length and orientation is the frame scale and orientation.

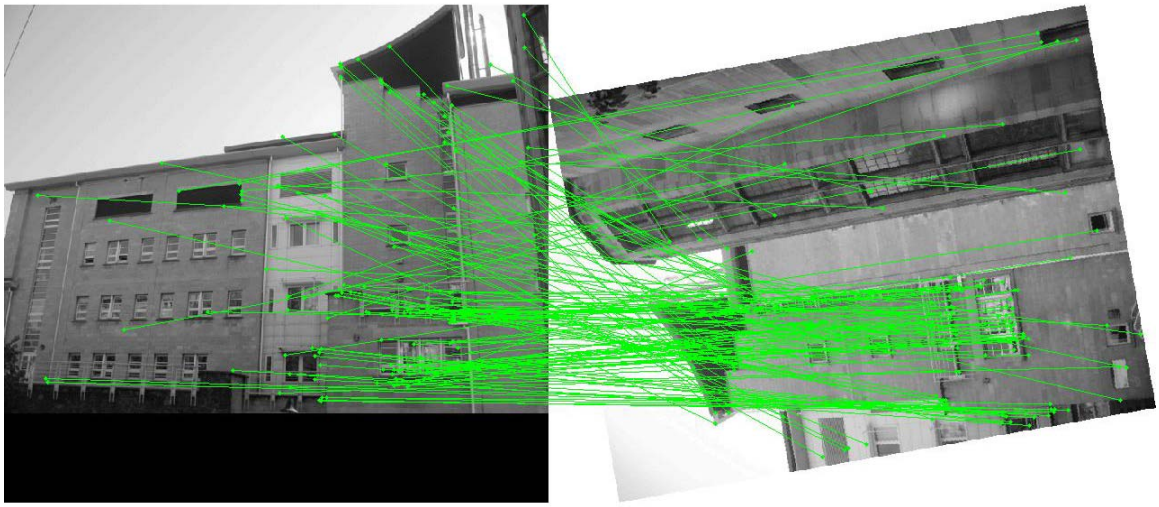

Figure 46(b): Shows the interconnection of points that MbC technique considers as putative matches between the two images. As it can be observed, there is a big set of wrong matches between the two images. In this step of the program, this was expected. The correlation window rotation option was enabled.

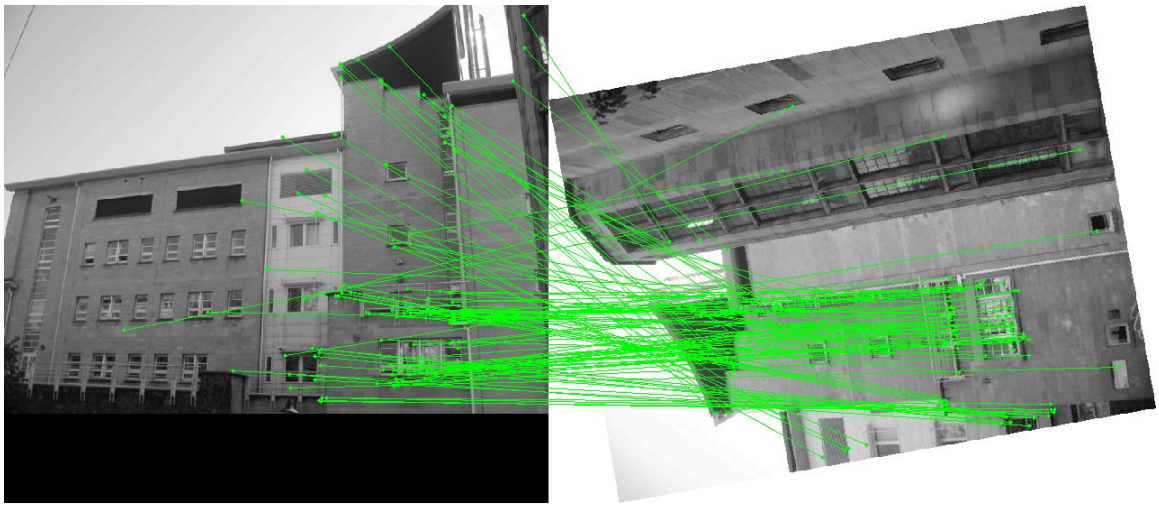

Figure 46(c): Shows the RANSAC algorithm results. The remaining points (from putative matches set), are the estimated inliers.

	Image 1 (Left)	Image 2 (Right)
SIFT – Frames detected	1646	2472
MbC – Putative matches	134	
RANSAC – Inliers estimation	107	
Matching Rate (MR)	80%	
Status	Images Matched	

Table 7: Shows the SIFT, Match by Correlation and RANSAC results for test 2.

	Execution Time (Seconds)
SIFT – Frames detected	0.007 s
MbC – Putative matches	153.879 s
RANSAC – Inliers estimation	0.179 s
Application Total Time	154.43 s (2.6 minutes)

Table 8: Shows the SIFT, Match by Correlation (MbC) and RANSAC execution time in seconds. The application total time includes any other function call.

4.1.2.2 Analysis

The results (tables 7, 8) show that the proposed application is invariant in rotation. The SIFT algorithm proved its invariance in rotation since a very big number of frames has been extracted. The correlation rotation option has been enabled in this example, in order to be able make the application invariant in rotation. The results show that the MbC technique is effective in rotation, however the application speed test shows that the MbC increases the overhead when this option is enable. The RANSAC algorithm discards a large amount of inliers, but three (3) of them are "escape" and considered as inliers.

4.1.3 Landscape Images with Different Viewpoint

In this example two landscape images were used of the same plan but with different image condition. These conditions involve a different view point, rotation and light scaling. This image is one of the images found in any paper related to object matching. Both of the images captured in a darker environment, thus to reduce the image brightness. The second image was captured with a 20° rotation and then re-rotated another 120°, thus to create a large difference between the orientation of the two images, see figure 47(a-c) Both images are converted to grayscale when the application starts, as it is explained in 3.1 (step1).The images properties are shown in table 9:

Properties	Image 1 (Left)	Image 2 (Right)
Format	JPEG	JPEG
Initial Size	600x480	600x480
Color	RGB	RGB

Table 9: Test 3 images properties.

4.1.3.1 Results

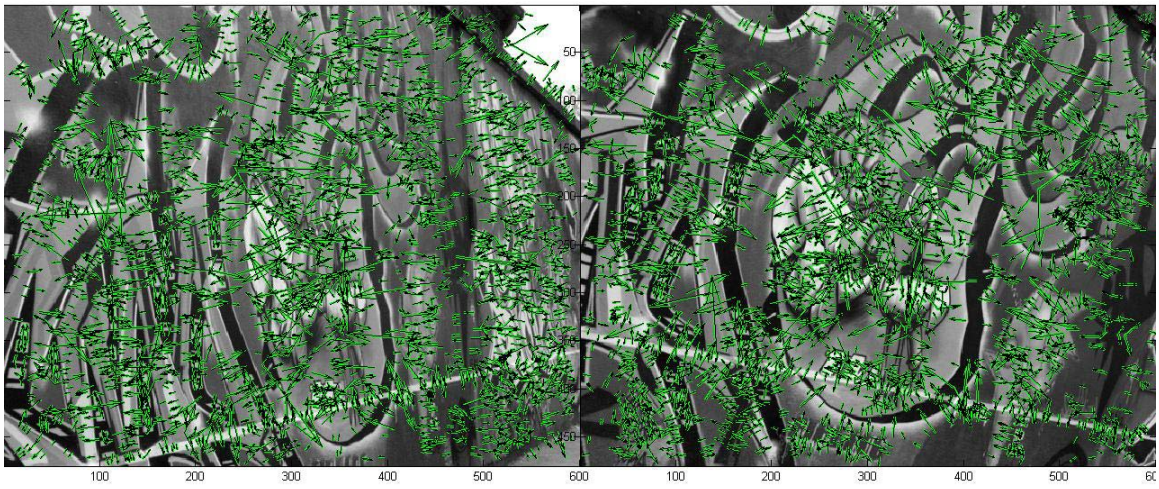

Figure 47(a): Shows the extracted SIFT frames for each image. The second image has a big difference in view point, about 10° rotation and light scaling. Because the second image is scaled, less number of SIFT frames are extracted, but it is still invariant under those conditions. Each arrow describes a SIFT frame, where the arrow length and orientation is the frame scale and orientation.

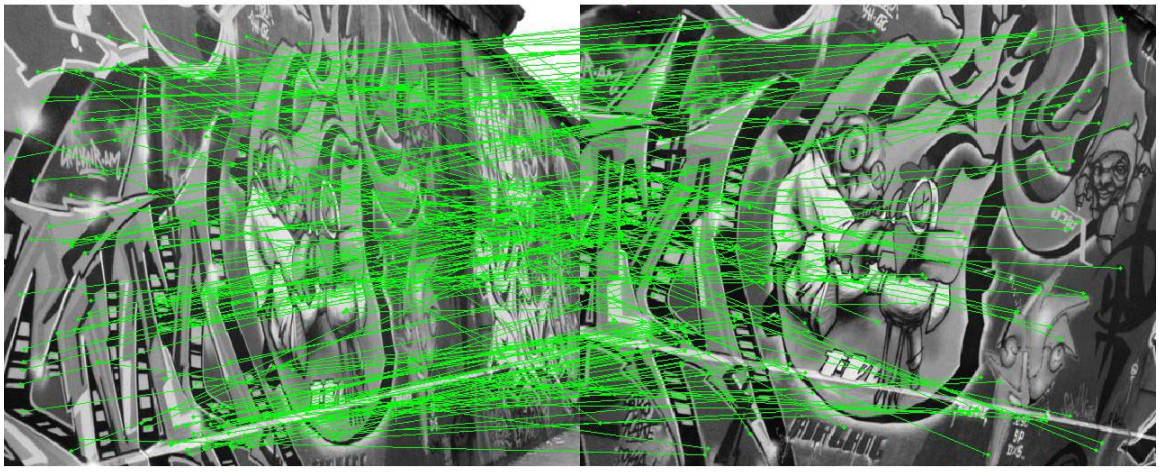

Figure 47(b): Shows the interconnection of points that MbC technique considers as putative matches between the two images. There is a big number of wrong matches between the two images, but at this step, this was expected.

Figure 47(c): Shows the RANSAC algorithm results. The remaining points (from putative matches set), are estimated as inliers. A big number of points was considered as outliers and were discarded.

	Image 1 (Left)	**Image 2 (Right)**
SIFT – Frames detected	3382	3073
MbC – Putative matches	193	
RANSAC – Inliers estimation	134	
Matching Rate (MR)	69%	
Status	Images Matched	

Table 10: Shows the SIFT, Match by Correlation and RANSAC results for test 3.

	Execution Time (Seconds)
SIFT – Frames detected	0.012 s
MbC – Putative matches	342.127 s
RANSAC – Inliers estimation	0.351 s
Application Total Time	342.70 s (5.7 minutes)

Table 11: Shows the SIFT, Match by Correlation (MbC) and RANSAC execution time in seconds. The application total time includes any other function call.

4.1.3.2 Analysis

The results (tables 10, 11) of these tests show that the proposed method can be also effective under the images conditioned as described in the test description. The SIFT algorithm continues to be invariant and fast. The big difference in the viewpoint including scaling and rotation did not affect the results. The MbC method with the invariance in rotation option enabled, shows that a large number of putative matches can be detected, but as the rotation invariance option is enabled and the SIFT frames number increases, the time overhead is increased dramatically. The RANSAC algorithm shows that it can detect the inliers, but also in this test due to the images background commonality, some outliers are considered as inliers.

4.2 Testing Using Real RS Company Components Images

In this part of the testing section, real images from electrical components provided by RS Electronics Company will be tested. A series of components' images was captured, thus to create a temporary database of components. This components database was used to check if the client component image (that was captured by him), match with any components from the database. The form of the database is a template image including a series of identical component images (smaller) from different perspectives, thus the client does not have any bounds and limitation in a specific point of view.
This part of the testing section is simulating how well the proposed application can be used for RS company requirements.

4.2.1 Defected Component Image VS Database

In this example, a defected component image (client image) captured in a low light environment, is compared with the identical database image. The defected component has bent and soldered pins. Database images are image captured that are considered "perfect" as this definition is introduced in section 3.2.4. *See figures48 (a-c)*. Both images are converted to grayscale when the application starts, as it is explained in 3.1 (step1).The images' properties are shown in table 12:

Properties	Image 1 (Left)	Image 2 (Right)
Format	TIF	TIF
Initial Size	288x216	223x206
Color	RGB	RGB

Table 12: Test 4 images properties.

4.2.1.1 Results

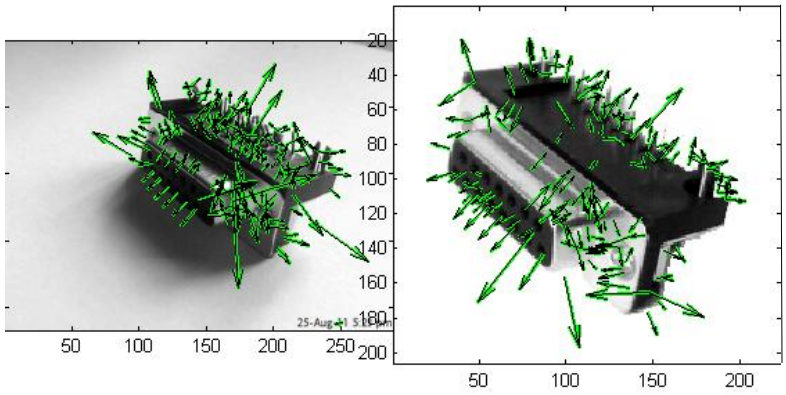

Figure 48(a): Shows the extracted SIFT frames for each image. Because the client images (left) is slightly defected and less bright, a lot of image details have been lost during SIFT frames extraction. Each arrow describes a SIFT frame, where the arrow length and orientation is the frame scale and orientation.

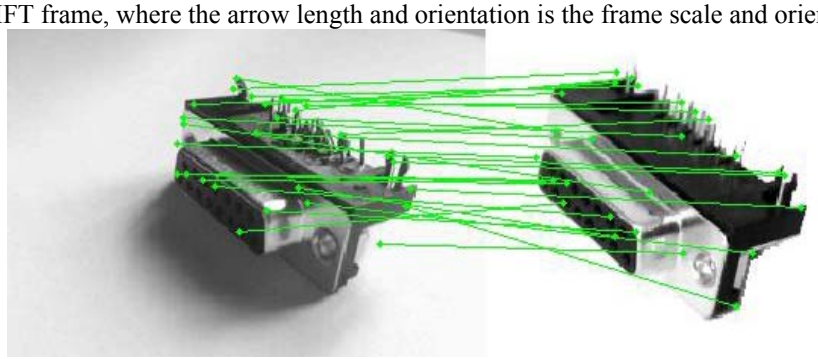

Figure 48(b): Shows the interconnection of points that MbC technique considers as putative matches between the two images. Because the components are identical, some wrong matches is expected.

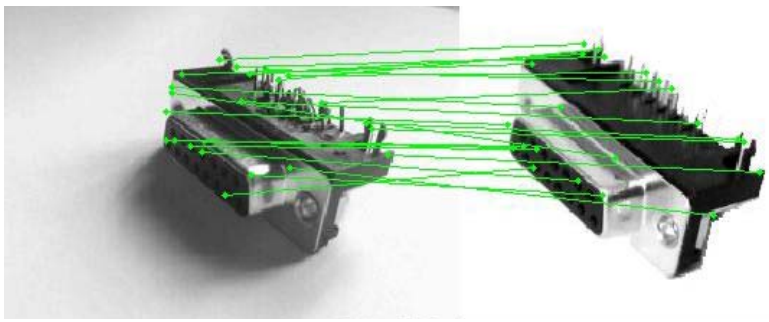

Figure 48(c): Shows the RANSAC algorithm results. The remaining points (from putative matches set), are estimated as inliers.

	Image 1 (Left)	Image 2 (Right)
SIFT – Frames detected	177	163
MbC – Putative matches	22	
RANSAC – Inliers estimation	27	
Matching Rate (MR)	81%	
Status	Objects Matched	

Table 13: Shows the SIFT, Match by Correlation and RANSAC results for test 4.

	Execution Time (Seconds)
SIFT – Frames detected	0.002 s
MbC – Putative matches	0.970 s
RANSAC – Inliers estimation	0.093 s
Application Total Time	1.17 s (0.00195 minutes)

Table 14: Shows the SIFT, Match by Correlation (MbC) and RANSAC execution time in seconds. The application total time includes any other function call.

4.2.1.1 Analysis

The results (tables 13, 14) of the tested components prove that as a first step the proposed method is effective when matching accuracy is required. The SIFT algorithm still extracts successfully image features even in the defective image. The MbC method responds to putative matches' detection. The correlation window rotation option is not required any more, since the image is tested within all possible perspectives within the database.
The RANSAC algorithm discards the outliers, with only three (3) to "escape" and be considered as inliers. The execution time result is fairly good, since only a second is needed for the object matching operation.

4.2.3 RS Company Electrical Almost Identical Components

In this example an object will be checked with itself from the record and with almost an identical one from the database. Both tests should succeed but it is expected the identical (left image) one to perform with higher matching rate than the almost identical. The difference between the two objects is that, one has more input terminals than the other. The pattern of the terminals is the same, thus making the final matching result more difficult. This is a very good test for application accuracy and performance, when almost identical objects are tested, figures 49(a-f). Both images are converted to grayscale when the application starts, as it is explained in 3.1 (step1). The images properties are shown in table 15:

Properties	Image 1 (Left Object)	Database 1 Image	Database 3 Image
Format	TIF	TIF	TIF
Initial Size	400x300	842x1191	595x842
Color	RGB	RGB	RGB

Table 15: Test 5 images properties.

4.2.3.1 Results

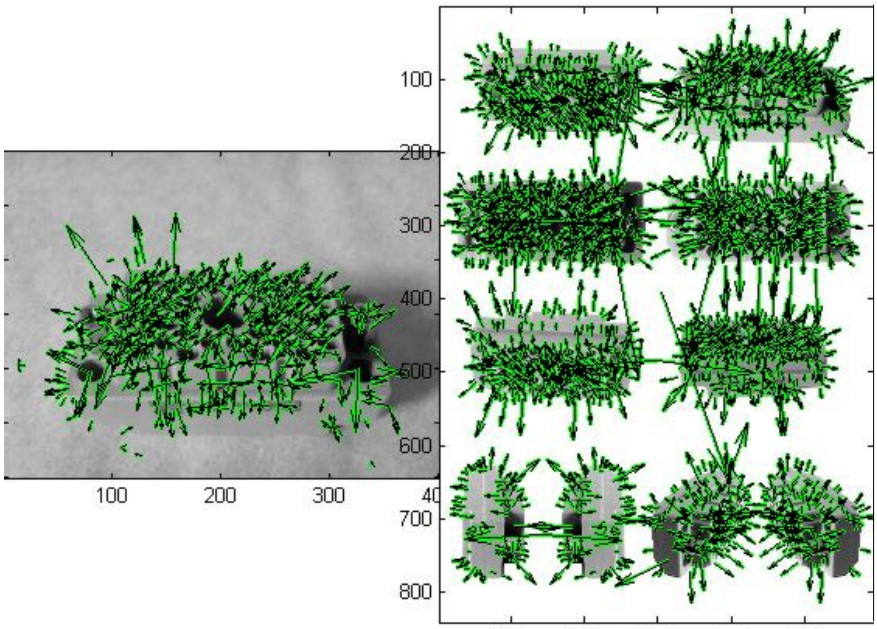

Figure 49(a): SIFT frames extraction for the completely identical objects

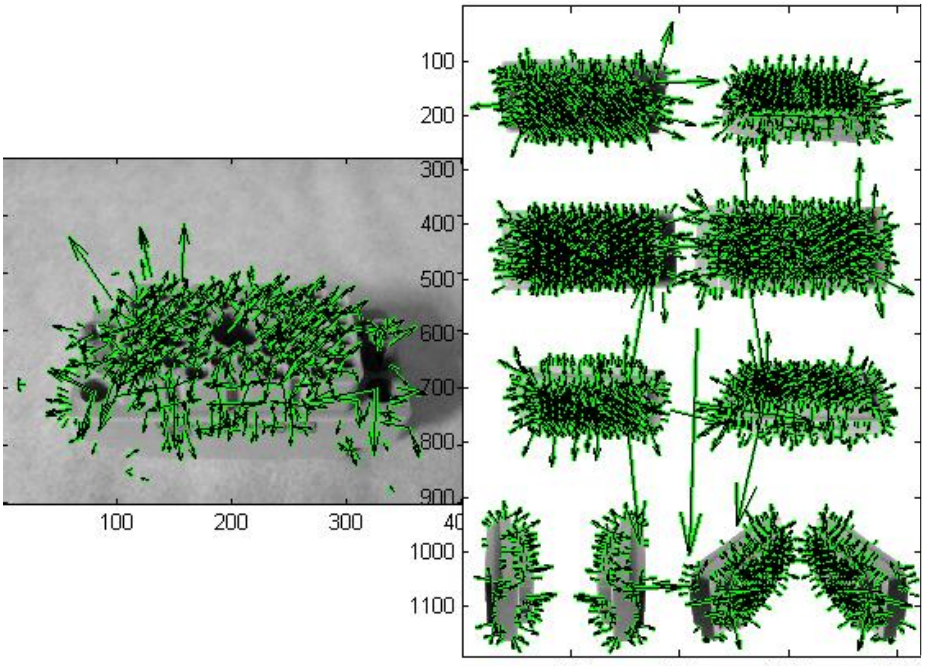

Figure 49(b): SIFT frames extraction for the almost identical objects

Figures 49(a, b): Shows the extracted SIFT frames for each image. The left image of each test is the same object used for both tests, so SIFT features number is the same. Because both databases (large images) include more than one object, a huge number of SIFT features is extracted for each one and because the object are almost identical some features properties are the same. Each arrow describes a SIFT frame, where the arrow length and orientation is the frame scale and orientation.

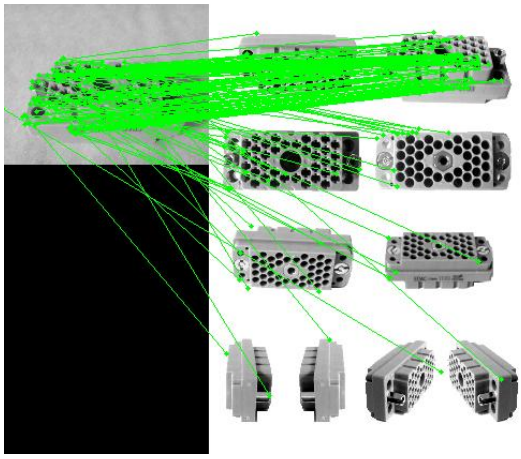

Figure 49(c): Putative matches for identical objects.

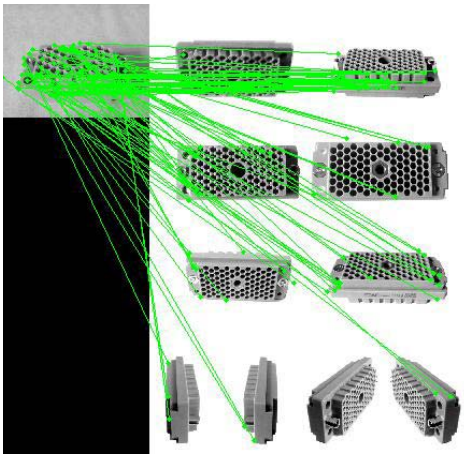

Figure 49(d): Putative matches for almost identical objects.

Figures(c, d): Shows the interconnection of points for each test where MbC technique consider as putative matches. There are some wrong matches between the two images, but at this step, this was expected.

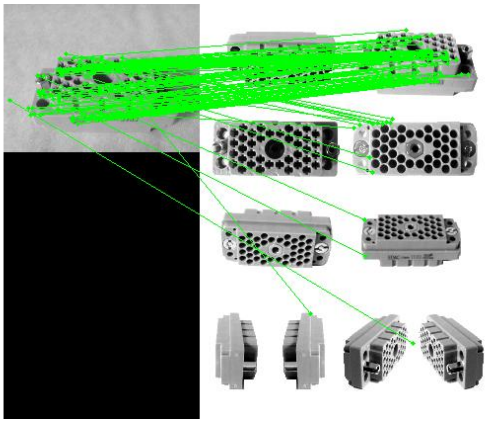

Figure 49(e): RANSAC inliers estimation for identical objects.

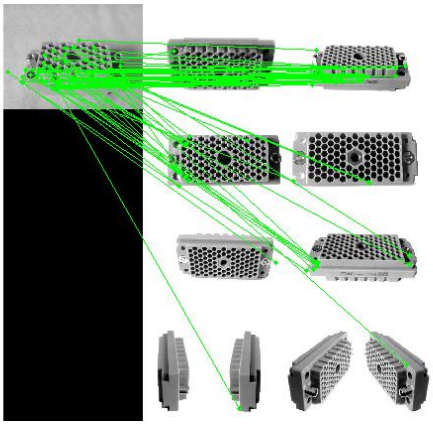

Figure 49(f): RANSAC inliers estimation for almost identical objects.

Figure 50(e, f): Shows the RANSAC algorithm results for both tests. The remaining points (from putative matches set) are estimated as inliers.

	Image 1 (Common)	Image Database 1	Image Database 2
SIFT – Frames detected	597	3265	6142
MbC – Putative matches		147	88
RANSAC – Inliers estimation		114	61
Matching Rate (MR)		77%	69%
Status		Objects Matched	Objects Matched

Table 16: Shows the SIFT, Match by Correlation and RANSAC results for test 5. The results under database 1 and 2 columns are relative to image 1, which is the examined object.

		Execution Time (Seconds)	
	Image 1 (Common)	Image 1 (Left) Database 1	Image 2 (Right) Database 2
SIFT – Frames detected		0.019 s	0.892 s
MbC – Putative matches		56.075 s	126.097 s
RANSAC – Inliers estimation		0.168 s	5.164
Application Total Time		56.43 s (0.9 minutes)	134.82 s (2.2 minutes)

Table 17: Shows the SIFT, Match by Correlation (MbC) and RANSAC execution time in seconds. The application total time includes any other function call for both tests. The results under database 1 and 2 columns are relative to image 1, which is the examined object.

4.2.3.2 Analysis

The results (tables 16,17) obtained from both tests verify the effectiveness of the application when almost identical objects are examined for their similarities. The SIFT algorithm extracts a large number of features, since in each database image eight (8) objects but in different perspective were included. Giventhe big number of extracted points, the SIFT algorithm execution time was less than 1 second. The MbC method executes the putative matches as it is expected but the large number of SIFT frames add extra latency in the execution, thus increasing the application execution time. Improvements in speed and how the overhead can be reduced will be explored in future work and in the improvements section.

5. Critical Evaluation

Through this section a critical evaluation of the project is explored. This is an analysis of the project objectives and whether or not the objectives have been achieved, as well if the choices that were made were the best and most successful.

A comparison of the proposed method with other object matching techniques will be performed, thus to evaluate the result of the proposed method based in this field area.

5.1 Project Objectives Evaluation

The primary aim of this project was to design and develop an effective application that will be able to decide whether two or more images match, calculate matching ratio and determine their common object. However, the project was split in two objectives.
The first objective was to define the collaboration between the algorithms selected from the research skills study, thus to be able to achieve the solution. This objective can be considered as successful and was achieved by finding the mathematical relation between the three algorithms. This relation was accomplished by deciding that a transformation between the input and the output of the algorithms must be performed. The output of one algorithm must be a valid input of the next. Regarding the SIFT algorithm, its output frames (keypoints) were stored in an array as an x-y coordinate points and then used by match by correlation (MbC) technique as input. Then MbC method calculates the putative matches between the two images and outputs this as points, which are then used by RANSAC algorithm, which estimated the inliers.

The second objective and most complicated was to replicate and optimise these algorithms, as well as to implement their collaboration in a program. Any algorithm optimisation that might be found necessary for the solution of any problem found during the implementation was also included in this objective. This objective was completed and can be also considered as successful as it met its requirement of the programming implementation of each algorithm. Each algorithm was operating in a different function designed in MATLAB and the proposed collaboration method defined in the first objective was accomplished. The correct collaboration between the three algorithms was very essential since the interchanged data should not change while moving between the algorithms. This sub-objective was accomplished successfully without any lost or distortion of data, thus the calculation of the matching rate at the end of the program was not affected. The application invariance in rotation, scaling, translation and change to view point requirement has been met by the correct algorithm implementation and collaboration. The rotation issue of MbC method was solved by implement a correlation window technique as it is explained in the implementation section.

The effectiveness of the proposed method was verified in the test section, where the results show that the second project objective, invariance to image different changes, has been met. The invariant to rotation issue that was detected in MbC was solved.
The results verify the proposed method performance with the average of succession rate up to 76%, between the tests. This result was obtained are illustrated in the graph in figure 5o.

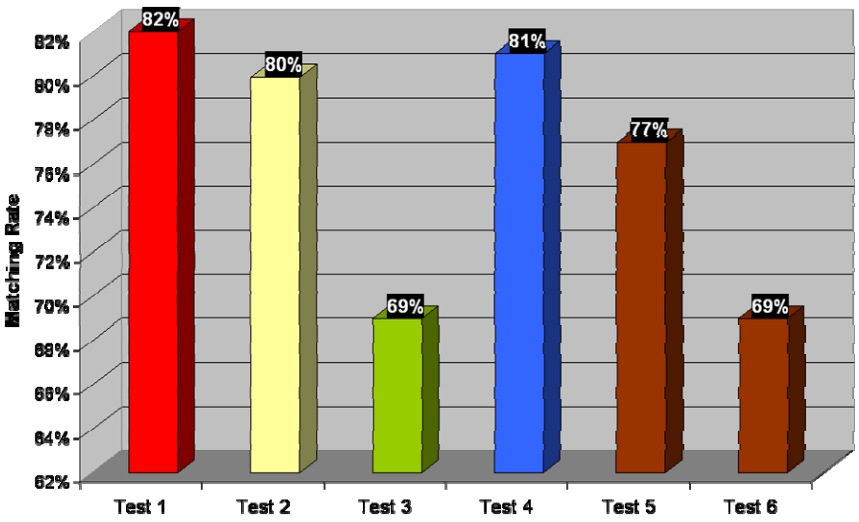

Figure 50: Shows the matching rate of each test with average of 76%.

In addition, when the application execution time is considered, there are some issues since the number of SIFT frames and the rotation invariance option increases the time overhead. However, the application efficiency can be improved and this is discussed in the next section. The application's total execution time for each test are illustrated in the graph of figure 51.

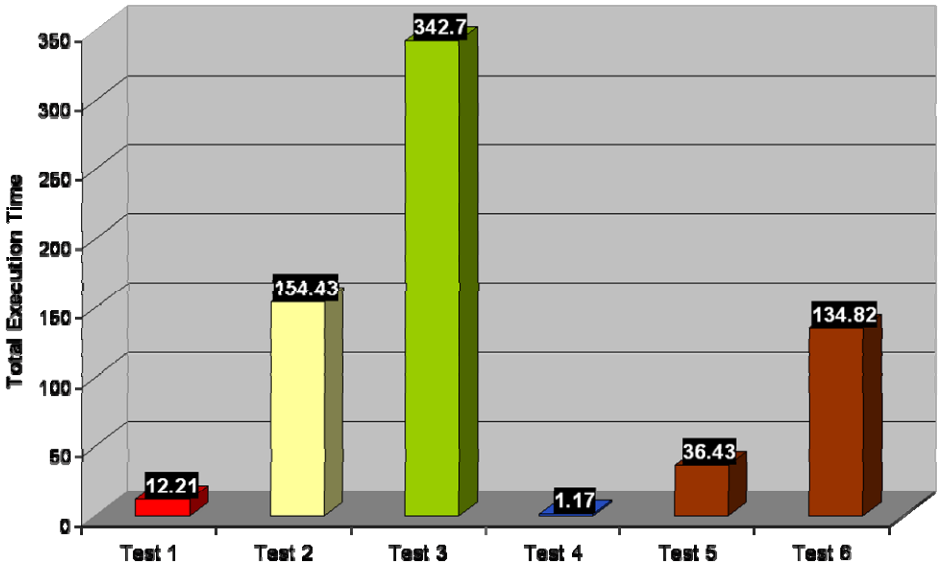

Figure 51: Shows the total execution time spend for each test.

5.2 Evaluating Comparing to Other Methods

In this section, the proposed method is compared with other object matching techniques. This comparison is performed to evaluate the performance of the application, thus to determine its competitiveness.

The image used in test 4.1.3, is from the public domain resources of INRIA and was used to compare the proposed method with the one proposed in [25]. The application was also tested over the following methods:

- Image matching by SIFT - Matching algorithm suggested by D.G. Lowe [1] and implemented by A. Vedali [33].
- Image matching by monogenic phase implemented by P. Kovesi [31].

These three methods are tested using the images from test 4.1.1 –to 5.2.3. The monogenic phase is a putative matches' detection method, so RANSAC was used for inliers estimation. The SIFT matching algorithm replace both MbC and RANSAC. For feature detections project, SIFT was used in all cases. The algorithm proposed in paper [25] is compared by using the results given in the publication paper for the image found in test 4.1.3. See table 18.

	Matched Points			
Test	**Proposed Method Inliers/MR**	**SIFT Matching [33]**	**Monogenic Phase [31] Inliers/MR**	**Normalized Cross-Correlation [25]**
4.1.1	62 / 82%	265	93 / 60%	
4.1.2	107 / 80%	441	190 / 54%	
4.1.3	134 / 69%	436	291 /42%	76
4.2.1	27 / 81%	12	36 / 66%	
4.2.3 (a)	114 / 77%	39	121/ 61%	
4.2.3 (a)	61 / 69%	39	111 / 57%	
Max. Time (Execution)	**134.82 s**	**2.72 s**	**136.51 s**	

Table 18: Shows the results of each algorithm in each test.

The monogenic phase algorithm detects much more putative matches than MbC but from the results one can observe that the matching rate when collaborating RANSAC is very low. Considering the speed test result is in the same level with the MbC, but MbC is much more effective.

When comparing the proposed method with SIFT matching we can see that SIFT matching algorithm detects more matched points. When it was tested in real components images its performance reduced dramatically, which means it is unstable. Also, it is very important to mention that SIFT matching algorithm allows one point to match more than one time, and this is the reason of the large number of points matched. Considering speed, it is much more efficient than the proposed method. The reason is, because it uses frames descriptor for distances calculation, thus reducing the interaction with the image which is adding more delay.

When the proposed method is compared with the normalised cross-correlation (NCC) method in the number of extracted points for the 4.1.3 test, the proposed method detects more points but observing the resulted images some of them are outliers. The results are not adequate to examine the two methods completely, but considering accuracy, the NCC method was more successful and considering detection skills, the proposed method is more successful.

6. Improvements and Future Work

Future work and improvements of the proposed method are explored in this section. The proposed method verifies its effectiveness when accuracy and performance are required. However, there is potential for improvements, especially when considering application efficiency. Future work and spending more time in the application is required to improve the performance and the efficiency, by exploring any other image processing techniques that may be found useful.

6.1 Improvements in Efficiency and Performance

By observing the tests, it is very clear that there is a delay in the MbC function execution. This delay reduction is crucial since the application is destined for a company ordering department. The company database holds thousands of images which means that it must be processed quickly. A way to reduce this delay and make the application fast is by using parallel connections., The application runs efficiently on a computer cluster or a grid of networked computer by using this toolbox, without changing any parts of the code [32]. Considering that a computer cluster can have up to 16 cores, it means that the application can run in 16 parallel connections, which will increase the process.

In addition, improvements in correlation window rotation option can increase the application efficiency and performance. The method can be re-considered and re-implemented thus to find a way to make it faster and more accurate. By analysing the MbC function, the parts of the code that produce most of the delay can be detected. The next step is to re-implement them considering performance and efficiency.

6.2 Future Work

The RS Company database contains components of real images as well as 3D graphical models for each image. After some initial tests of the SIFT algorithm in 3D graphical models, the results were very optimistic, since SIFT frames are extracted from all the detail points of the components, thus creating "perfect" samples. Further tests involving the whole application will help to explore all the possible cases for any different approach in object matching image processing. *See figure 52.*

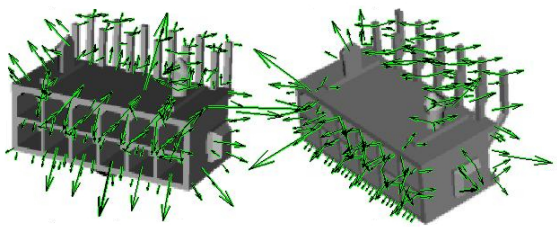

Figure 52: Shows SIFT frames extracted from 3D graphical models.

7. Conclusions

A new method is presented for matching two uncalibrated images. The method employs rotation and scale invariance, extracting and detecting distinctive features from images. Rotation and scaling invariance issues were solved. The application is implemented in MATLAB R2010a and tested in a single core PC. Image captured environment is affecting the application performance and efficiency. When more features are extracted and rotation affects the testing image, the total execution time is increased. Experimental results on real RS company electrical connectors images, verify the effectiveness of the application with respect of accuracy and performance. By comparing the proposed method with other object matching techniques it was observed that it can be very robust against its opponents and in some case more successful, especially when accuracy is the judgment criterion. There is however, potential for improvements which were proposed in future work.

8. References

[1] Lowe, D. (2004) Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

[2] Bakken, P T. (2007) An evaluation of the SIFT, algorithm for CBIR. Fornebu, Telenor Research and Innovation. (R&I N 30/2007.)

[3] Hasler, D., Sviaz, L., Susstrunk, S., Vetterli, M. (2003) Outlier modeling in image matching. PAMI, IEEE Trans., vol. 25, no. 3, pp. 301–315.

[4] Duda, R. O., and Hart, P. E. (1972) Use of the hough transformation to detect lines and curves in pictures. Communications of the ACM, vol. 15, no. 1, pp. 11–15.

[5] Ballard D. (1981) Generalizing the hough transform to detect arbitrary shapes. Pattern Recognition, vol. 13, no. 2, pp. 111–122.

[6] Duerig, T. (2006) *Scale Invariant Feature Transform*. cseweb.ucsd.edu/~tduerig/SIFT.ppt [Accessed 31st August 2011].

[7] Estrada, F., Jepson, A., Fleet, D. (2004) *Local Features Tutorial*. http://www.cs.toronto.edu/~jepson/csc2503/tutSIFT04.pdf [Accessed 31st August 2011].

[8] Bishop, C.M. (1992) Exact calculation of the Hessian matrix for the multilayer perceptron. Neural Computation 4 (4), 494.

[9] Fischler, M., and Bolles., R. (1981) Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. Commun. ACM., 24:381–395.

[10] Hartley, R. and Zisserman, A. (2002) Multiple view and geometry in computer vision. Cambridge, University Press.

[11] Zuliani, M. (2009) RANSAC for DUMMIES. Vision Research Lab University of California Santa Barbara. http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf [Accessed 31st August 2011]

[12] Derpanis, G. K. (2010) *Overview of the RANSAC Algorithm Version 1.2*, York University, Computer Science Ph.D, Summary Paper.

[13] Estrada F.J., Jepson A.D., Fleet D. (2004). *Planar Homographies.* Lecture notes, http://www.cs.utoronto.ca/~strider/vis-notes/tutHomography04.pdf [Accessed 31st August 2011].

[14] Van Belle, G., and Millard, S. P. (1998) *STRUTS: Statistical rules of thumb.* Seattle, WA: The Northwest Research Center for Statistics and the Environment. http://www.nrcse.washington.edu/NRCSE/struts/chapter2.pdf [Accessed 31st August 2011]

[15] Moisan, L. and Stival, B. (2004) Aprobabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix. International Journal on ComputerVision, 57(3), pp201–218.

[16] Rabin, J., Delon, J., Gousseau, Y., Moisan, L. (2010) MAC-RANSAC: a robust algorithm for the recognition of multiple objects. In Fifth International Symposium on 3D Data Processing, Visualization and Transmission.

[17] Jaechul, K. (2008) *Camera parameter estimation for image based modeling*. Demo presentation,Visual recognition and search. http://www.cs.utexas.edu/~grauman/courses/spring2008/slides/Jaechul-Camera_parameter_estimation_for_Image_Based_Modeling.pdf [Accessed 31st August 2011]

[18] (2009) RANdom SAmple Consensus (RANSAC) algorithm. http://dsin.blogspot.com/2009/07/random-sample-consensus-ransac.html [Accessed 31st August 2011]

[19] Abdi, H., Edelman, B., Valentin, D., Dowling, W.J., (2009) Experimental design and analysis for psychology. Oxford University Press, Oxford.

[20] Cheatham, L. M. () A Practical Guide to Statistics. http://www.surgicalcriticalcare.net/resources.php [Accessed 31st August 2011]

[21] Higgins, J. (2005) The Radical Statistician by Jim Higgins. Chapter 2: The Correlation Coefficient. http://www.spanglefish.com/adamsonuniversity/documents/Psychological%Research/CORRELATION%20COEFFICIENT.pdf [Accessed 31st August 2011]

[22] Gerstman, B. (2007)StatPrimer: 14. Correlation. San Jose State University, 14 March 2007 [Lecture Notes Taken by Burt Gerstman]. http://www.sjsu.edu/faculty/gerstman/StatPrimer/correlation.pdf [Accessed 31st August 2011]

[23] http://www.mathworks.co.uk/help/toolbox/images/ref/corr2.html

[24] Stahlberg, H., Zumbrunn, R., Engel, A. (2002) Digital Image Processing in Natural Sciences and Medicine. http://www.c-cina.unibas.ch/teaching/uc-davis/dip-2002/dip-0-contents.pdf [Accessed 31st August 2011]

[25] Zhao, F., Huang, Q., Gao, W. (2006) Image Matching By Normalized Cross Correlation. Proceedings of International Conference on Acoustics, Speech and Signal Processing, vol. 2, pp. 729–732.

[26] Lewis, J. P. (1995) Fast normalized cross-correlation. In Vision Interface.

[27] Smith, S. W. (2006) *The Scientist and Engineer's Guide to Digital Signal Processing*, San Diego, CA: California Technical Publishing.

[28] http://www.mathworks.co.uk/products/parallel-computing/index.html l

[29] Hartley, R. and Zisserman, A. (2000) *Multiple view geometry in computer vision*, Cambridge University Press: Cambridge, UK.

[30] Vedaldi, A. (2006) siftmatching.m. The Regents of the University of California. UCLA Vision Lab, Department of Computer Science.

[31] Kovesi, P. (2005) Monogenicphase.m. School of Computer Science & Software Engineering, The University of Western Australia.

[32] http://www.mathworks.co.uk/products/parallel-computing/index.html

[33] Allen, C. R. (2006) The British industrial revolution in global perspective: how commerce created the industrial revolution and modern economic growth. http://www.nuff.ox.ac.uk/users/allen/unpublished/econinvent-3.pdf [Accessed 31st August 2011]

[34] Vedaldi, A. (2006) An implementation of SIFT detector and descriptor University of California. UCLA Vision Lab, Department of Computer Science.

[35] P.H.S. Torr, A. Zisserman, S.J. Maybank (1995) Robust detection of degenerate configurations for the fundamental matrix, iccv, pp.1037, Fifth International Conference on Computer Vision (ICCV'95)

*All figures created manually by the author, using the relevant designing tools, except the figures that include a reference.

**All illustrations are in high quality. Changes in quality may occur because of doc to pdf conversion.

9. Appendix: Source code

projectRS.m – Main Application file

```
function projectRS(im1,im2)
    contime=0;
    step=1;
    fprintf('\n------------RS Project------------\n') ;
    fprintf('%d.Loading images...\n',step); tic; step=step+1;
    if nargin == 0

        %%APPLICATION TESTING

        im1 =imreadbw('data/dif3.tif');im2=imreadbw('data/Samples/B20opt2.tif');

        %If image is very big
       % [x,y]=size(im1); im1=imresize(im1,[x/3,y/3]);
       % [x,y]=size(im2); im2=imresize(im2,[x/3,y/3]);
    end
    fprintf('  Images loading completed. Time: %.3f sec\n',toc);
    contime=contime+toc;

    %Properties selection
    matchmethod=3; %Method of Matching / mBc=3
    printIndices=1; %Printing SIFT Results, 1=Yes 0=No
    %---End of Properties

    im1c=im1;
    im2c=im2;

    im1=imsmooth(im1,.1) ;
    im2=imsmooth(im2,.1) ;

    im1=im1-min(im1(:)) ;
    im1=im1/max(im1(:)) ;
    im2=im2-min(im2(:)) ;
    im2=im2/max(im2(:)) ;

    fprintf('%d.SIFT operation started...\n',step); tic;step=step+1;
    S=3;
    %Descriptors are 128 because the 4x4=16 => 16*8(8 directions)= 128
    [frames1,descr1,gss1,dogss1] = sift( im1, 'Verbosity', 1, 'Threshold',0.009,
'NumLevels', S ,'boundarypoint', 1) ;
    [frames2,descr2,gss2,dogss2] = sift( im2, 'Verbosity', 1, 'Threshold',0.009,
'NumLevels', S ,'boundarypoint', 1) ;
     sifT=toc;
    fprintf('  SIFT operation completed. Time: %.3f sec\n',toc) ;
    contime=contime+toc;

    %Printing indices of descriptors for each image
    %printIndices=1;
    if printIndices

         figure('Name','Extracted Features'); clf;
        tightsubplot(1,2,1) ; imagesc(im1) ; colormap gray ; axis image ;
        hold on ;
        h=plotsiftframe( frames1 ) ; set(h,'LineWidth',2,'Color','g') ;
        h=plotsiftframe( frames1 ) ; set(h,'LineWidth',1,'Color','k') ;

        tightsubplot(1,2,2) ; imagesc(im2) ; colormap gray ; axis image ;
        hold on ;
        h=plotsiftframe( frames2 ) ; set(h,'LineWidth',2,'Color','g') ;
        h=plotsiftframe( frames2 ) ; set(h,'LineWidth',1,'Color','k') ;

    end
    fprintf('  -> Image 1 # of descriptors: %d \n', numel(descr1)/128) ;
```

```
    fprintf('  -> Image 2 # of descriptors: %d \n', numel(descr2)/128) ;


    %Rounding of array "a" & "b" elements
    a=frames1(1:2,:); %a=a+1;
    a=round(a);

    b=frames2(1:2,:); %b=b+1;
    b=round(b);
    %--End of Rounding

    %Reverse frames to row to column
    ac(1,:)=a(2,:);
    ac(2,:)=a(1,:);
    bc(1,:)=b(2,:);
    bc(2,:)=b(1,:);
    %--End of reversing rows-to-columns


    fprintf('%d.Matching operation started...\n',step); tic; step=step+1;
    contToRansac=1; %Continue to RANSAC flag initialisation
    %matchmethod=2;

                fprintf('  -> Correlation Match algorithm is running...\n');
                d1=numel(im1);
                d2=numel(im2);
                dmax=max(d1,d2); dmax=sqrt(dmax); dmax=dmax/3; dmax=round(dmax);
                dmax = inf;      % Maximum search distance for matching
                w = 23;        % Window size for correlation matching
                fprintf('**** Maximum Distance: %d | Window Size: %d\n',dmax,w);
                % Use normalised correlation matching
                [m1,m2, indices1, indices2, matches] = MbC(im1 , ac , im2 , bc , w,
dmax);
                % Assemble homogeneous feature coordinates for fitting of the
                % fundamental matrix, note that [x,y] corresponds to [col, row]
                putative1 = [m1(2,:); m1(1,:); ones(1,length(m1))];
                putative2 = [m2(2,:); m2(1,:); ones(1,length(m1))];


    if (numel(putative1(1,:))<=7 || numel(putative2(1,:))<=7)
        fprintf('  Not enough number of points found.\n  ->Result: OBJECTS NOT
MATCHED\n');
        return;
    end
     mbcT=toc;
    fprintf('  Matching operation completed. Time: %.3f sec\n',toc) ;
    contime=contime+toc;
    %--End of Matching process


     fprintf('%d.RANSAC operation started...',step); tic; step=step+1;

    if contToRansac

        t = 0.1;

        % 8 points fundamental matrix solutions,
        [F, inliers] = ransacfitfundmatrix(putative1, putative2, t, 0);

        fprintf('  -> Number of inliers: %d (%d%%) \n', ...
                length(inliers),round(100*length(inliers)/length(m1)))
                fprintf('  -> Number of theoretical matches: %d \n', length(m1))
                 ranT=toc;
                fprintf('  RANSAC operation completed. Time: %.3f sec\n',toc);
                contime=contime+toc;

        l2 = F*putative1;    % Epipolar lines in image2
        l1 = F'*putative2;   % Epipolar lines in image1

        % Solve for epipoles
        [U,D,V] = svd(F);
```

```
        e1 = hnormalise(V(:,3));
        e2 = hnormalise(U(:,3));
    else

        fprintf('  \nRANSAC operation cannot completed!  \n-> ERROR: No putatuve matches
found. Time: %.3f sec\n',toc);
    end

    if contToRansac
        figure('Name','Putative matches'); clf;
        plotmatches(im1c,im2c,a,b,matches,'Stacking','h','interactive',0) ;
        inlmatches=inliersTomatches(inliers,putative1,putative2);
        figure('Name','Inliers matches'); clf;

plotmatches(im1c,im2c,putative1,putative2,inlmatches,'Stacking','h','interactive',0) ;
        drawnow; %or a % b if matches(1,:)=indices1;
    end

    fprintf('  ->Result: OBJECTS MATCHED\n');
    fprintf('%d.Exit project. Total time: %.2f sec %.1f
minutes\n',step,contime,contime/60);
    fprintf('----------------END---------------\n');
```

SIFT.m – Extract SIFT frames

```
function [frames,descr,GSS,DoG]=SIFT(I,varargin)

if(nargin < 1)
  error('At least one argument is required.') ;
end

[M,N,C] = size(I) ;

% Lowe's equivalents choices
S      =  3 ; omin   = -1 ; O  = floor(log2(min(M,N)))-omin-3 ; % up to 8x8 images
sigma0 = 1.6*2^(1/S) ;                  % smooth lev. -1 at 1.6
sigman = 0.5 ; thresh = 0.04 / S / 2 ;
r      = 10 ; NBP    = 4 ;
NBO    = 8 ;magnif = 3.0 ;

% Parese input
compute_descriptor = 0 ;
discard_boundary_points = 1 ;
verb = 0 ;

for k=1:2:length(varargin)
  switch lower(varargin{k})

    case 'numoctaves'
      O = varargin{k+1} ;

    case 'firstoctave'
      omin = varargin{k+1} ;

    case 'numlevels'
      S = varargin{k+1} ;

    case 'sigma0'
      sigma0 = varargin{k+1} ;

    case 'sigman'
      sigmaN = varargin{k+1} ;

    case 'threshold'
      thresh = varargin{k+1} ;

    case 'edgethreshold'
      r = varargin{k+1} ;
```

```
    case 'boundarypoint'
     discard_boundary_points = varargin{k+1} ;

    case 'numspatialbins'
      NBP = varargin{k+1} ;

    case 'numorientbins'
      NBO = varargin{k+1} ;

    case 'magnif'
      magnif = varargin{k+1} ;

    case 'verbosity'
     verb = varargin{k+1} ;

    otherwise
      error(['Unknown parameter ''' varargin{k} '''.']) ;
  end
end

% Arguments sanity check
if C > 1
  error('I should be a grayscale image') ;
end

frames      = [] ;
descr = [] ;

% Compute scale spaces

GSS = gaussianss(I,sigman,O,S,omin,-1,S+1,sigma0) ;
DoG = diffss(GSS) ;


for o=1:GSS.O

  % Local maxima of the DOG octave
  % The 80% tricks discards early very weak points before refinement.
  idx = siftlocalmax(DoG.octave{o}, 0.8*thresh) ;
  idx = [idx , siftlocalmax( - DoG.octave{o}, 0.8*thresh)] ;

  K=length(idx) ;
  [i,j,s] = ind2sub( size( DoG.octave{o} ), idx ) ;

 y=i-1 ;
 x=j-1 ;
  s=s-1+DoG.smin ;
  oframes = [x(:)';y(:)';s(:)'] ;

  % Remove points too close to the boundary
  if discard_boundary_points

    rad = magnif * GSS.sigma0 * 2.^(oframes(3,:)/GSS.S) * NBP / 2 ;
    sel=find(...
      oframes(1,:)-rad >= 1                    & ...
      oframes(1,:)+rad <= size(GSS.octave{o},2) & ...
      oframes(2,:)-rad >= 1                    & ...
      oframes(2,:)+rad <= size(GSS.octave{o},1)     ) ;
    oframes=oframes(:,sel) ;

  end

  % Refine the location, threshold strength and remove points on edges
  oframes = siftrefinemx(...
    oframes, ...
    DoG.octave{o}, ...
    DoG.smin, ...
    thresh, ...
    r) ;
```

```
  % Compute the oritentations
  oframes = siftormx(...
    oframes, ...
    GSS.octave{o}, ...
    GSS.S, ...
    GSS.smin, ...
    GSS.sigma0 ) ;

  % Store frames
  x     = 2^(o-1+GSS.omin) * oframes(1,:) ;
  y     = 2^(o-1+GSS.omin) * oframes(2,:) ;
  sigma = 2^(o-1+GSS.omin) * GSS.sigma0 * 2.^(oframes(3,:)/GSS.S) ;
  frames = [frames, [x(:)' ; y(:)' ; sigma(:)' ; oframes(4,:)] ] ;

  % descr
  if nargout > 1

    sh = siftdescriptor(...
      GSS.octave{o}, ...
      oframes, ...
      GSS.sigma0, ...
      GSS.S, ...
      GSS.smin, ...
      'Magnif', magnif, ...
      'NumSpatialBins', NBP, ...
      'NumOrientBins', NBO) ;

    descr = [descr, sh] ;

  end

end
```

MbC.m – Match by Correlation

```
function [m1, m2, indices1, indices2, matches, corlmap] = ...
                  MbC(img1, putative1, img2, putative2, w, maxdist)

if nargin == 5
    maxdist = Inf;
end
img1c=img1; img2c=img2;
img1 = double(img1);
img2 = double(img2);

%Contrast Enhansing
img1 = img1 - filter2(fspecial('average',w),img1);
img2 = img2 - filter2(fspecial('average',w),img2);

% Generate correlation matrix
corlmap = CorrelationMap(img1, putative1, img2, putative2, w, maxdist);
[corrows,corcols] = size(corlmap);

% Find max along rows give strongest match in putative2 for each putative1
[mputative2forputative1, colputative2forputative1] = max(corlmap,[],2);

% Find max down cols give strongest match in putative1 for each putative2
[mputative1forputative2, rowputative1forputative2] = max(corlmap,[],1);

% Now find matches that were consistent in both directions
indices1 = zeros(1,length(putative1));  % Arrays for storing matched indices
indices2 = zeros(1,length(putative2));
indcount = 0;
for n = 1:corrows
        check=rowputative1forputative2(colputative2forputative1(n));
        %check2=rowputative1forputative2(colputative2forputative1(corrows-n+1));
       % check2=colputative2forputative1(rowputative1forputative2(n));
        if  check == n% consistent both ways
                indcount = indcount + 1;
                indices1(indcount) = n;
                indices2(indcount) = colputative2forputative1(n);
        end
end

% Trim arrays of indices of matched points
indices1 = indices1(1:indcount);
indices2 = indices2(1:indcount);

% Extract matched points from original arrays
m1 = putative1(:,indices1);
m2 = putative2(:,indices2);

matches(1,:)=indices1;
matches(2,:)=indices2;

%---------------------------------------------------------------------
% Here correlation map is computed

function corlmap = CorrelationMap(img1, putative1, img2, putative2, w, maxdist)

    if mod(w, 2) == 0
        error('Window size should be odd');
    end

    [rows1, npts1] = size(putative1);
    [rows2, npts2] = size(putative2);

    % Initialize correlation matrix values to -infinty
     corlmap=-ones(npts1,npts2)*Inf;
    %{
    for i=1:10
            xcorlmap{i}=-ones(npts1,npts2)*Inf;
     end
```

```
    %}
    % END-- Initializing correlation matrix values to -infinty

    if rows1 ~= 2 | rows2 ~= 2
        error('Feature points must be specified in 2xN arrays');
    end

    [img1rows, img1cols] = size(img1);
    [img2rows, img2cols] = size(img2);

    wbig=(w*2)-1;
    rbig = (wbig-1)/2;
    r = (w-1)/2;   % 'radius' of correlation window
    %rbig = ((w*2)-2)/2;


    % Find indices of points that are distance 'r' or greater from
    % boundary on image1 and image2;

    n1ind = find(putative1(1,:)>r & putative1(1,:)<img1rows+1-r & ...
        putative1(2,:)>r & putative1(2,:)<img1cols+1-r);

    n2ind = find(putative2(1,:)>r & putative2(1,:)<img2rows+1-r & ...
        putative2(2,:)>r & putative2(2,:)<img2cols+1-r);

    for i=1:10 ; WFS{i}=0;  end

    for n1 = n1ind
        % Generate window in 1st image

        w1 = img1(putative1(1,n1)-r:putative1(1,n1)+r, putative1(2,n1)-
r:putative1(2,n1)+r);
        %w1big = img1(putative1(1,n1)-rbig:putative1(1,n1)+rbig, putative1(2,n1)-
rbig:putative1(2,n1)+rbig);

        %w1=imrotate(w1,10,'crop');

        %{
        degrees=0;

        for i=1:10
            %img1TMP=imrotate(img1,degrees,'crop');
            wtmp=imrotate(w1big,degrees,'crop');
            window{i} = imrotate(w1,degrees,'crop');
            w1crop{i}=imcrop(wtmp,[12,12,22,22]);
            degrees=degrees+20;
        end

        %}
        % Pre-normalise w1 to a unit vector.

        %for i=1:10
            %window{i}=window{i}./sqrt(sum(sum(window{i}.*window{i})));
        %end


        w1 = w1./sqrt(sum(sum(w1.*w1)));
        %w1rot = w1rpt./sqrt(sum(sum(w1rot.*w1rot)));


        % Identify the indices of points in putative2 that we need to consider.
        if maxdist == inf
            n2indmod = n2ind; % We have to consider all of n2ind

        else     % Compute distances from putative1(:,n1) to all available putative2.
            putative1pad = repmat(putative1(:,n1),1,length(n2ind));
            dists2 = sum((putative1pad-putative2(:,n2ind)).^2);
            %Find indices of points in putative2 that are within distance maxdist of
putative1(:,n1)
            n2indmod = n2ind(find(dists2 < maxdist^2));
        end
```

```
        % Calculate normalised correlation measure.  Note this gives
        % significantly better matches than the unnormalised one.

        for n2 = n2indmod
            % Generate window in 2nd image
            w2 = img2(putative2(1,n2)-r:putative2(1,n2)+r, putative2(2,n2)-
r:putative2(2,n2)+r);

            %{
            for i=1:10
                rcoef = corr2(w1crop{i},w2);
                if rcoef >=0.70 %< min
                    WFS{i}=WFS{i}+1;

                end
            end

           %}
            corlmap(n1,n2) = sum(sum(w1.*w2))/sqrt(sum(sum(w2.*w2)));
            %corlmap(n1,n2) = sum(sum(window{pos}.*w2))/sqrt(sum(sum(w2.*w2)));

           % for i=1:10
           %     xcorlmap{i}(n1,n2)=sum(sum(window{i}.*w2))/sqrt(sum(sum(w2.*w2)));
           % end

        end
    end


%  corlmap=xcorlmap{pos};

return
```

RANSAC.m – Estimate inliers

```
% RANSACFITFUNDMATRIX - fits fundamental matrix using RANSAC

% Copyright (c) 2004-2005 Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% http://www.csse.uwa.edu.au

% February 2004  Original version
% August   2005  Distance error function changed to match changes in RANSAC
% June 2011 Remove unnecessary code parts

function [F, inliers] = FRANSAC(x1, x2, t, feedback)

    if ~all(size(x1)==size(x2))
        error('Data sets x1 and x2 must have the same dimension');
    end

    if nargin == 3
    feedback = 0;
    end

    [rows,npts] = size(x1);
    if rows~=2 && rows~=3
        error('x1 and x2 must have 2 or 3 rows');
    end

    if rows == 2    % Pad data with homogeneous scale factor of 1
        x1 = [x1; ones(1,npts)];
        x2 = [x2; ones(1,npts)];
    end

    % Normalise each set of points so that the origin is at centroid and
```

```
    % mean distance from origin is sqrt(2).
    [x1, T1] = normalise2dpts(x1);
    [x2, T2] = normalise2dpts(x2);

    s = 8;

    fittingfn = @fundmatrix;
    distfn    = @funddist;
    degenfn   = @isdegenerate;
    [F, inliers] = ransac([x1; x2], fittingfn, distfn, degenfn, s, t);
    F = fundmatrix(x1(:,inliers), x2(:,inliers));

    % Denormalise
    F = T2'*F*T1;

%--------------------------------------------------------------------------
function [bestInliers, bestF] = funddist(F, x, t);

    x1 = x(1:3,:);    % Extract x1 and x2 from x
    x2 = x(4:6,:);


    if iscell(F)  % We have several solutions each of which must be tested

    nF = length(F);   % Number of solutions to test
    bestF = F{1};     % Initial allocation of best solution
    ninliers = 0;     % Number of inliers

    for k = 1:nF
        x2tFx1 = zeros(1,length(x1));
        for n = 1:length(x1)
        x2tFx1(n) = x2(:,n)'*F{k}*x1(:,n);
        end

        Fx1 = F{k}*x1;
        Ftx2 = F{k}'*x2;

        % Evaluate distances
        d =  x2tFx1.^2 ./ ...
         (Fx1(1,:).^2 + Fx1(2,:).^2 + Ftx2(1,:).^2 + Ftx2(2,:).^2);

        inliers = find(abs(d) < t);     % Indices of inlying points

        if length(inliers) > ninliers   % Record best solution
        ninliers = length(inliers);
        bestF = F{k};
        bestInliers = inliers;
        end
    end

    else     % We just have one solution
    x2tFx1 = zeros(1,length(x1));
    for n = 1:length(x1)
        x2tFx1(n) = x2(:,n)'*F*x1(:,n);
    end

    Fx1 = F*x1;
    Ftx2 = F'*x2;

    % Evaluate distances
    d =  x2tFx1.^2 ./ ...
         (Fx1(1,:).^2 + Fx1(2,:).^2 + Ftx2(1,:).^2 + Ftx2(2,:).^2);

    bestInliers = find(abs(d) < t);     % Indices of inlying points
    bestF = F;                          % Copy F directly to bestF

    end


%--------------------------------------------------------------------------
% (Degenerate!) function to determine if a set of matched points will result
```

```
function r = isdegenerate(x)
    r = 0;
```