5-2013

Identifying Robust SIFT Features for Improved Image Alignment

Sanjay Abhinav Vemuri
University of Arkansas, Fayetteville

IDENTIFYING ROBUST SIFT FEATURES
FOR IMPROVED IMAGE ALIGNMENT

IDENTIFYING ROBUST SIFT FEATURES FOR IMPROVED IMAGE ALIGNMENT

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

By

Sanjay Abhinav Vemuri
Manipal University
Bachelor of Science in Computer Science, 2009

May 2013
University of Arkansas

ABSTRACT

Several modern feature detection algorithms use Gaussian scale spaces in order to locate scale-invariant and rotationally invariant feature points in an image, including the Scale-Invariant Feature Transform (SIFT) algorithm. These SIFT features are used to enable a wide range of applications, including object recognition, motion tracking, and image stitching. One problem with SIFT is the fact that the number of features detected in an image can become very large, especially if the input image is big or has a lot of detail. This slows down feature matching and reduces the performance of these applications.

In this thesis, we will study different ways to improve feature matching by increasing the quality and reducing the number of SIFT features. We created an algorithm to identify robust SIFT features by evaluating how invariant individual feature points are to changes in scale. This allows us to exclude poor SIFT feature points from the matching process and obtain better matching results in reduced time. We also developed techniques consider scale ratios and changes in object orientation when performing feature matching. This allows us to exclude false-positive feature matches and obtain better image alignment results.

This thesis is approved for recommendation
to the Graduate Council.

Thesis Director:

_____

Dr. John Gauch

Thesis Committee:

_____

Dr. Craig Thompson

_____

Dr. Christophe Bobda

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _____
Sanjay Abhinav Vemuri

Refused _____
Sanjay Abhinav Vemuri

ACKNOWLEDGEMENTS

I would like to offer my sincere thanks to Dr. John Gauch for his guidance, help and support throughout my academic career.

I would also like to express my thanks and appreciation for the support my parents have offered me, and for the love of learning and science they have instilled in me.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In computer vision and image processing, feature detection is a process of finding unique interesting points in an image. In our research we use a technique called SIFT (Scale Invariant Feature transform) that was created by David Lowe to detect and extract features in an image that are invariant to transformations such as scale, rotation and illumination. SIFT has been used in a wide range of applications object recognition, motion tracking, image matching and image stitching.

SIFT has three stages, feature detection, feature extraction and key point reduction. In feature detection important feature points in an image are located. These feature points are found at different levels of Gaussian blurring of an image so we can say they have different scales. In feature extraction, we find 128 feature descriptors that describe a feature. The input of feature extraction algorithm is the neighborhood of the feature, and the output of this algorithm is a reduced representation of the input called as feature vector. Feature vector is like the fingerprint of the feature that describes edge orientation information. The third phase of SIFT is reduction of features which is also the final stage. In this stage, some of the feature points that lie on the edges or have low contrast are removed.

Even though features calculated are reduced, they still can be large in number especially when the image size is big. When the features are large in number that can be from few hundreds to thousands, they become cumbersome for an application that uses them for further processing. One such case would be feature matching. Feature matching is commonly used in image matching, motion tracking or object recognition. In image matching when two images are being matched first, features for both the images are calculated then a matching algorithm is applied to both the set of features from two images. Here every feature has to be processed but

only some of the features actually get matched by the algorithm unless both the images are identical. In order to speed up the matching, we need to remove some features that do not participate in the feature matching.

The goal of our research is to find and remove bad features from the overall set of SIFT features for a given image. Bad features are generally not robust or scale invariant thus they do not participate in any kind of matching more often. When bad features are removed the number of SIFT features significantly decrease but it does not affect the actual matching much.

There are two stages to our algorithm, first is the preprocessing and next is post processing. In the preprocessing we process the input image to remove unnecessary SIFT features and keep the good SIFT features. We do this by tracking SIFT feature points through a sequence of interpolated images, and counting the number of times a point occurs in this sequence. Points that occur very few times are not scale invariant, while points that are found in most of the interpolated images are very scale invariant.

In the post processing we do the actual matching between two preprocessed images. Here we also show how feature matching can be improved by considering scale ratio and orientation windows of the matched features. Even though preprocessing takes a while to finish, post processing time is greatly reduced. If we use the same preprocessed input image in matching with a number of different images, then matching becomes fast. This is very useful in the case of object matching in a video where one input image is matched to many images or all the frames of the video.

The rest of this document is organized as follows. Chapter 2 describes SIFT in more detail and related work in feature detection. In Chapter 3 we describe the design and implementation of our approach. Chapter 4 outlines our testing environment and Chapter 5

describes our results using representative input images. Finally, Chapters 6 and 7 contain our conclusions and future work.

2. BACKGROUND AND RELATED WORK

In this chapter we explain how feature detection algorithm works. Background concepts and definitions related to our research are explained with examples. In 2.1 and 2.2 we will give an overview of feature detection and extraction. In 2.3 and 2.4 we will discuss scale space theory and spatial filtering. In 2.5 we will discuss rotational invariance. Finally, in 2.6 and 2.7 we discuss in detail how SIFT and feature matching work.

2.1 Feature Detection

In computer vision and image processing, feature detection is used to find a part of an image which is unique or interesting. These feature points can be used to identify objects in an image or to solve a computational task related to an image. The computational task might be to recognize an object in the image or match one image to another.

Features are extracted generally by neighborhood operations applied on an image. Features can be of different structures like points, edges, regions etc. Some feature types can be easy to calculate but generally may not be good enough for future use. Different kinds of features work well for different types of scenarios but features that are mostly used are feature points. Feature points are good because they work well in most of the computer vision techniques. Feature detection can be computationally expensive and even using those features for any other purpose can be computationally costly.

Edges are curves or lines or set of points that show the intensity transitions in an image. Edges are easy to calculate but are difficult to track if the image content is changed slightly. For features like edges we use detection techniques which are based on gradients, Laplacian zero crossing etc. Image derivatives are generally used to locate edges in an image. Gradient is a first

derivative of an image and is used in edge detection. The gradient of an image gives us the direction and magnitude of maximum intensity change at every pixel of an image. First derivative of an image is calculated as

$$\nabla f(x,y) = (\partial f/\partial x, \partial f/\partial y) = (f_x, f_y)$$

Gradient direction is given by $\theta = \tan^{-1}(f_y/f_x)$ and gradient magnitude is given by

$$|\nabla f| = \sqrt{{f_x}^2 + {f_y}^2}$$

Gradients are estimated by calculating partial derivatives at each pixel. Partial derivatives are calculated using convolution masks dx and dy.

$$f_x(x,y) = f(x,y) * dx(x,y)$$

$$f_y(x,y) = f(x,y) * dy(x,y)$$

Different kinds of masks like two point estimate, Roberts cross, and Sobel can be used for convolution. The 3x3 Sobel masks for dx and dy are given by:

$$dx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad dy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Laplacian zero crossing is another edge detection method. In this case, we find the zero crossings of the Laplacian. The Laplacian is defined as

$$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = f_{xx} + f_{yy}$$

Laplacian is estimated by calculating partial derivatives at each pixel. Partial derivatives are calculated by convolving masks dxx and dyy since Laplacian is second order derivative.

One of the common masks used for calculating Laplacian is $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

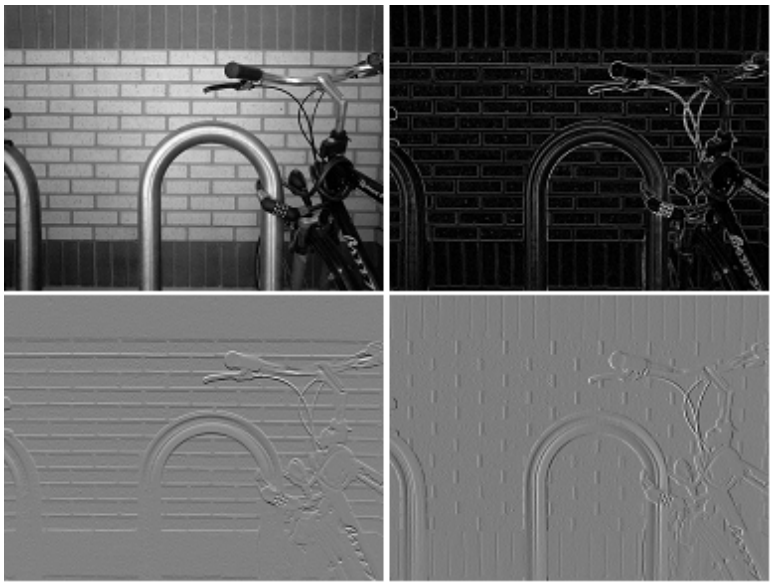

Figure 1 - Top-left: original image, top-right: gradient magnitude from Sobel operator, bottom-left: y-gradient from Sobel operator, bottom-right: x-gradient from Sobel operator [11]

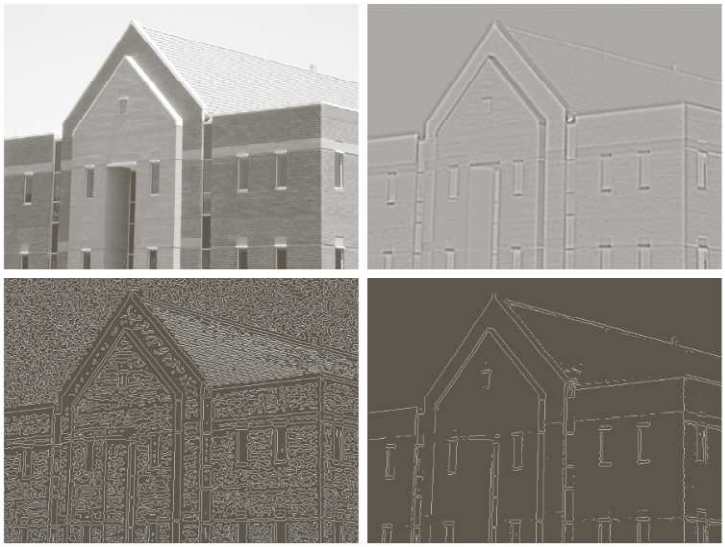

Figure 2 - Top-left: original image, top-right: image convolved with Laplacian operator, bottom-left: zero crossings of Laplacian image, bottom-right: zero crossings with threshold applied [5]

2.2 Feature Extraction

Once the feature point has been detected, the local image patch around that feature is extracted. The local neighborhood of the feature might consist of a lot of information so this information is processed using an efficient algorithm. The output of this algorithm is generally a reduced representation of the local neighborhood of the feature detected. The output is typically called as feature descriptors or feature vector, and this should be easy to use in the future. Feature detection and extraction might need a lot of image processing since the number of features in an image and amount of local neighborhood can be very large.

2.3 Scale Space

Objects in the real world have a certain size and shape but the size varies, depending on the distance from which we are viewing them. Scale space theory tries to replicate this concept with images. To find features that are scale invariant we use scale space theory on images. If a feature detection algorithm is applied to an image at different scales or sizes it should be able to produce features that are the same and this makes those features scale invariant. Suppose an object is viewed from a close distance, details of that object are clear and fine but as we go farther away from the object the detail becomes blurred. This concept can be applied to images using a smoothing or a blurring function. The most effective and commonly used smoothing function is Gaussian filter since it is almost similar to natural eyes function. If an image is scaled up we have greater detail and if it is scaled down we have less detail. This phenomenon can be produced by blurring that image in an increasing order. Here the scale varies as the smoothing filter size varies. So scale invariant features are found at different scales or Gaussian filter sizes.

An image is taken and blurred increasingly to produce a set of images. Each image is blurred with a different filter size. So the scale of an image is equal to the degree of blur used for that case. Now that a set of blurred images are created we use them to find features that are scale invariant. When processing any image for features we generate scale space of that image and find scale invariant features so the scale of the input image being used does not make a difference.

Figure 3 - The same scene smoothed to varying degrees, representing progressively smaller scales of the image [3]

2.4 Spatial Filtering

Spatial filters use fixed sized neighborhood in an input image to calculate output intensity. Gaussian filter uses Gaussian function to define the neighborhood weights. These weights are convolved with the neighborhood pixel values to get the blurred intensity values for that neighborhood.

Gaussian smoothing removes noise and unwanted details in an image. The blur level of Gaussian filter is determined mostly by the value of sigma. The amount of blurring increases along with the value of sigma. The weight at each pixel is given by

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

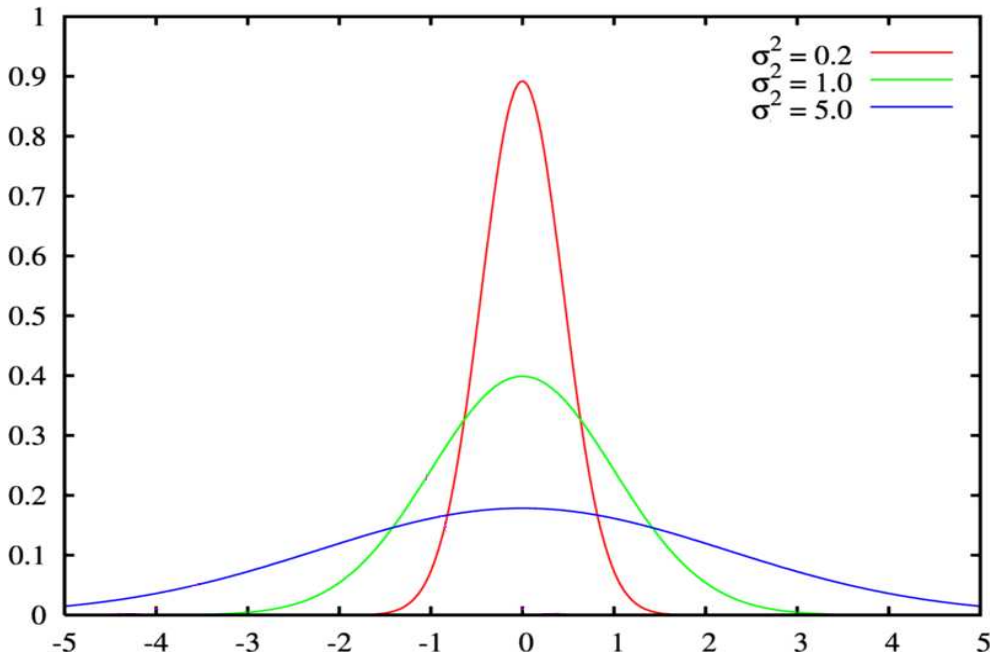

Figure 4 - The effects of varying sigma on a Gaussian curve

2.5 Rotational Invariance

The goal of most feature detection algorithms is to obtain features which are rotationally invariant. In other words, the features we get in an image should be unchanged if we rotate the image and calculate features again. There are two basic approaches to do this:

First, pick features we know are rotationally invariant based on their mathematical properties. For example, gradient magnitude of an image $|\nabla f| = \sqrt{{f_x}^2 + {f_y}^2}$ remains the same

even after the image is rotated. Even the Laplacian of an image $\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = f_{xx} + f_{yy}$ is rotationally invariant. In both the cases, we need to align the gradient magnitude or the Laplacian of the original image and the rotated image, for them to match.

Second, calculate some preferred direction for each feature and calculate every other direction in a feature based on this preferred direction. For example, In SIFT features calculated are given an orientation of the prominent gradient direction and all the remaining feature descriptors are calculated based on this prominent gradient direction. This solves the alignment issue and the SIFT features become rotationally invariant.

2.6 SIFT

SIFT is one of the most common algorithms used to detect and extract features. It was published by David Lowe in 1999. SIFT feature vectors are invariant to image transformations like translation, scaling, rotation and are also partially invariant to affine distortion and illumination change.

In SIFT an image is taken and smoothed several times to create a scale space. Then the original image is interpolated or scaled to half its original size then smoothed again progressively. This is repeated until the image is reduced to the smallest size. The smoothing function used here is Gaussian blur operator. Now we have different sets of images called octaves. Each octave is a set of images with same size that are blurred increasingly. These blurred images or octaves are used to produce another set of images called difference of Gaussian images (DoG). The DoG images are a result of difference between consecutive

Gaussian images in the scale space or octaves. These Difference of Gaussian images provide a good approximation to the Laplacian of an image [10].

The Laplacian of an image is calculated by blurring the image to remove the noise and then taking the second order derivative of that image. The Laplacian of an image is rotationally invariant. The Laplacian at (x, y) in an image is the same as Laplacian at the rotated coordinate say (x', y') in the rotated image. Since calculating Laplacian of images is computationally time consuming, which is why the DOG approximation is widely used.

We use the DoG images to find maxima and minima of the Laplacian. This produces a large number of feature points. Some of these feature points lie on the edge or have low contrast so removing them would be a good idea. If intensity of a key point in actual image is less than a certain threshold we consider it as a low contrast key point and discard it.

Feature points that lie on the edges are removed. Two gradients at the key point are calculated and these gradients are perpendicular to each other. We keep the feature points or corners where both the gradients are big. The feature points that are created are scale invariant. Features that are detected have different scales or Gaussian blur factor.

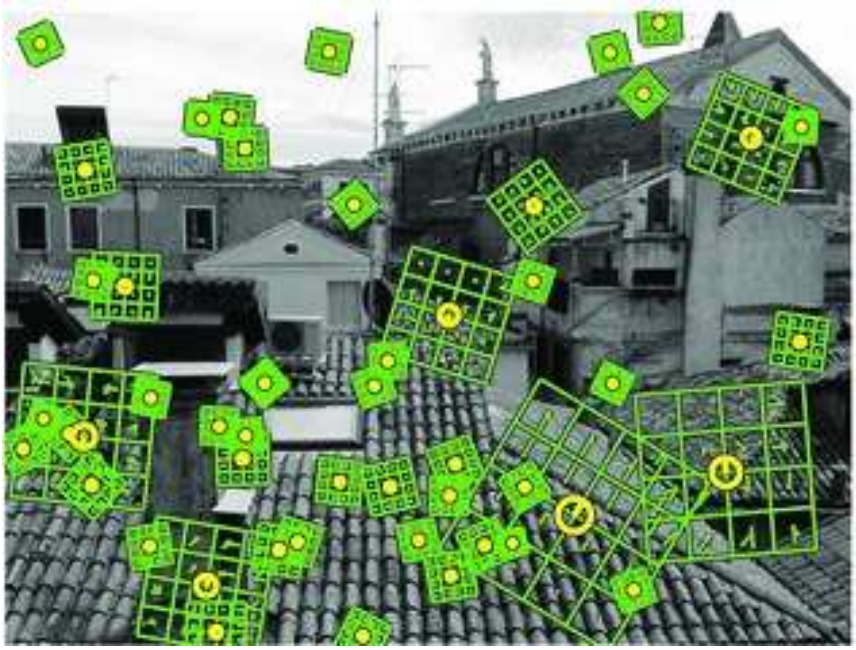

Figure 5 - Example of few feature points obtained with VLFeat SIFT [7]

The goal of feature extraction is to process the image to obtain geometric or intensity features from the neighborhood of each feature point. This collection of values is called a feature vector or feature descriptor. The feature vector makes use of local orientation information so it is rotationally invariant. The idea is to collect gradient directions and magnitudes around each key point. Then figure out the most prominent orientation in that neighborhood. This orientation is saved with the key point. The remaining calculations make use of this prominent gradient direction to ensure that the feature vector is rotationally invariant. Gradient directions in the neighborhood are used to create 128 feature descriptors. Finally a SIFT feature has a location (x, y) and a Gaussian scale factor (or the blur level) at which it was found. Hence a SIFT feature can be defined as vector $= [x, y, s, \theta, f1, f2, f3 \ldots f128]$ where s is scale, θ is the orientation angle and f1, f2…f128 are feature descriptors.

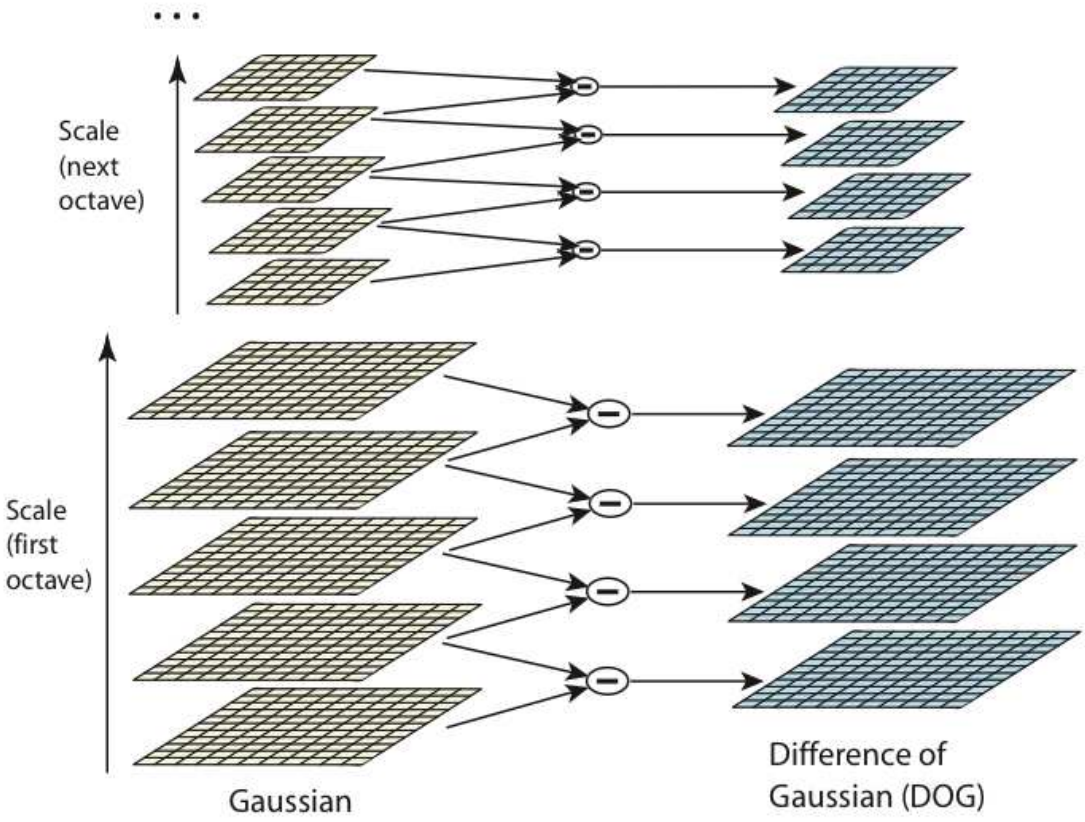

Figure 6 - Composition of Gaussian Scale Space [2]

2.7 Feature Matching

Features matching can be used to support a wide range of applications. For example, matching the features from a known object to features in an image can be used for object recognition. A similar approach can be used to align overlapping image to stitch together a panorama. In this section we discuss the issues of matching features to each other. Given a feature f1 in input image I1 we have to calculate the best match say feature f2 in input image I2. In brute force method of matching, every feature in I1 is paired with every feature in I2 and distance between the feature vectors of each pair is calculated in order to find all possible matches. The distance function compares each feature in I1 with f2 in I2 and finds a feature f2,

where distance between f1 and f2 is the least. In some cases f1 might not even have a good match f2. To eliminate these matches we set a threshold for the minimum distance. In most of the applications, distance is calculated as the sum of squared differences of the 128 descriptors between the feature vectors.

When this feature comparison is only done from I1 to I2 (or from I2 to I1) we may have incorrect matches. These false positives occur because we are trying to find best match literally for every feature in I1 or I2. To solve this, we can perform feature matching both ways, that is f2 is the best for f1 and f1 is the best for f2 and the distance between vectors of f1, f2 is the least among all the matches in both the cases of matching from I1 to I2 and I2 to I1. This greatly reduces false positives but we may also remove a small number of true positives.

In order to remove the false positives in the matching result and to make it more meaningful, we use an algorithm called RANSAC (random sample consensus). Input to this algorithm is a data set that is assumed to have a mathematical model whose parameters are unknown. This input data is also assumed to have some inliers that are true positives. The inliers present in the input satisfy a particular mathematical model. The goal of the algorithm is to find the inliers and discard the outliers if any. We assume that there are a certain minimum amount of inliers in the given input dataset.

For example, if we consider mean square line fitting where a set of points are given and we need to find a line equation that fits in best for all the points. Here we use all the points in the data set both inliers and outliers. If we use RANSAC we can improve the line fitting by calculating the line equation using the inliers and removing the outliers from the data set. This is illustrated in Figure 7.

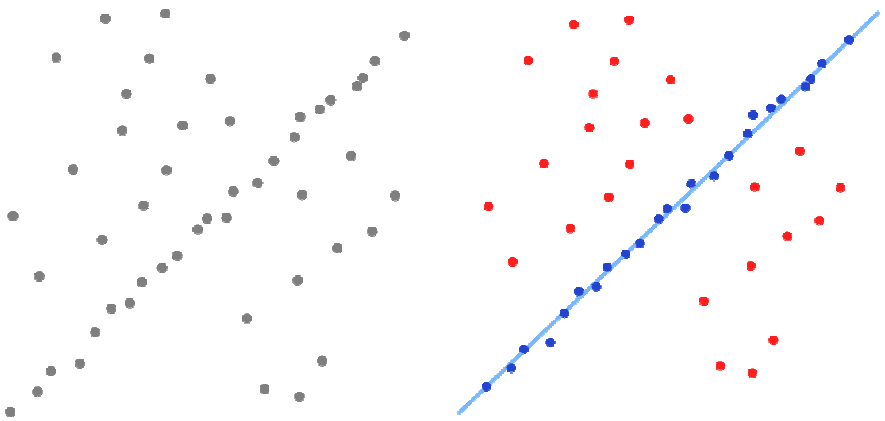

Figure 7 - left: A dataset with many inliers and outliers to which a line has to be fitted, right: Fitted line with RANSAC using all the inliers(blue), outliers(red) have no influence on the result [11]

First, some points in the data set are chosen randomly and assumed to be inliers and then a line equation is calculated based on these inliers. The remaining points in the data set are checked for inliers if any point is found to be an inlier then it is added to the hypothetical inlier set and the line equation is recalculated based on this inlier set. This goes on until all the points are divided into inliers and outliers. If the number of inliers is less than the previous result, the whole process is repeated by assuming new inliers. The algorithm has fixed number of iterations for assuming different inlier set. The algorithm chooses the best line equation by finding the case with the most inliers.

In the case of images, when feature points in image I1 match feature points in I2 we can calculate the rotation, translation, scaling necessary to align these two images. This transformation is called a homography. In this case RANSAC can be used to solve for the 3x3 homography matrix, by finding the transformation with the largest number of feature match pairs (inliers).

3. APPROACH

In this section we present different techniques to improve feature matching. In section 3.1 we discuss how to create a scale ratio window to improve feature matching. In section 3.2, we discuss how to create an orientation angle difference window. In section 3.3, we discuss how to create a count for each feature, based on how many times it is found in the interpolated copies. In sections 3.4 and 3.5 we evaluate the use of counts for SIFT and SURF feature matching. In section 3.6 we discuss how to reduce descriptor operations during matching using a scale test.

3.1 Scale Ratio

Even after using a good feature matching algorithm, the result may contain some false positives. In order to choose the correct matches we create a window for the correct scale ratio. Each SIFT feature has a scale assigned to it when it was detected. This scale value corresponds to the level of blurring applied to the input image when the feature point was detected. Hence large objects in the image will have large scales (they were detected after a lot of blurring) and small objects will have small scales.

When features in one image are matched to features in another image each feature might have a different scale but there ratio of scales should be constant. The scale ratio is also close to the ratio of sizes of the images being matched. If a feature f1 with a scale s1 matches a feature f2 with a scale of s2 then s1/s2 or s2/s1 is constant.

In reality, the scale ratios are not exactly constant but are somewhat close to each other, within a certain range. If we can get a rough estimate of this range we can then remove certain outliers as bad matches. All the scale ratios of the matching features generally fall into a small range of values. This range increases slightly when the image being matched is skewed.

In order to automatically separate inliers from outliers, we have devised an algorithm to window matches based on scale ratios. After using the brute force matching algorithm, we calculate the scale ratios for each feature match. Then we calculated mean, median and standard deviation for all the scale ratios. We use this information to define a scale ratio window as [mean-(factor)*stddev, mean+(factor)*stddev] or [median-(factor)*stddev, median+(factor)*stddev]. The value of the factor decides the size of the window. When the window is small we tend to lose some good matches along with the bad ones but this might save us some time while estimating homography.

We illustrate this algorithm below. In Figure 9, we show a plot of the scale ratio for 61 points that were found using brute force feature matching. Notice how there is a band of points in the range [1.5…2]. In Figure 10, we show the 41 points that fall in our scale window.

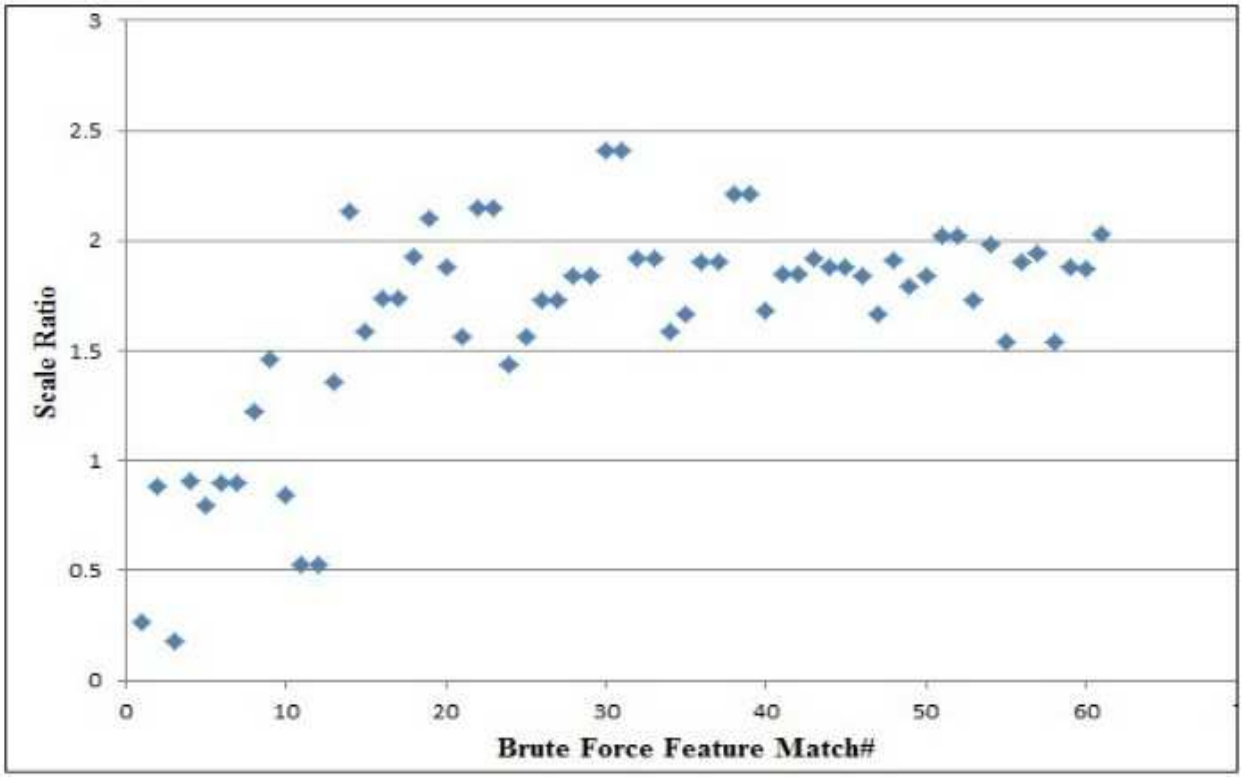

Figure 8 - Scale ratios of the matches found using Brute Force Matching Algorithm

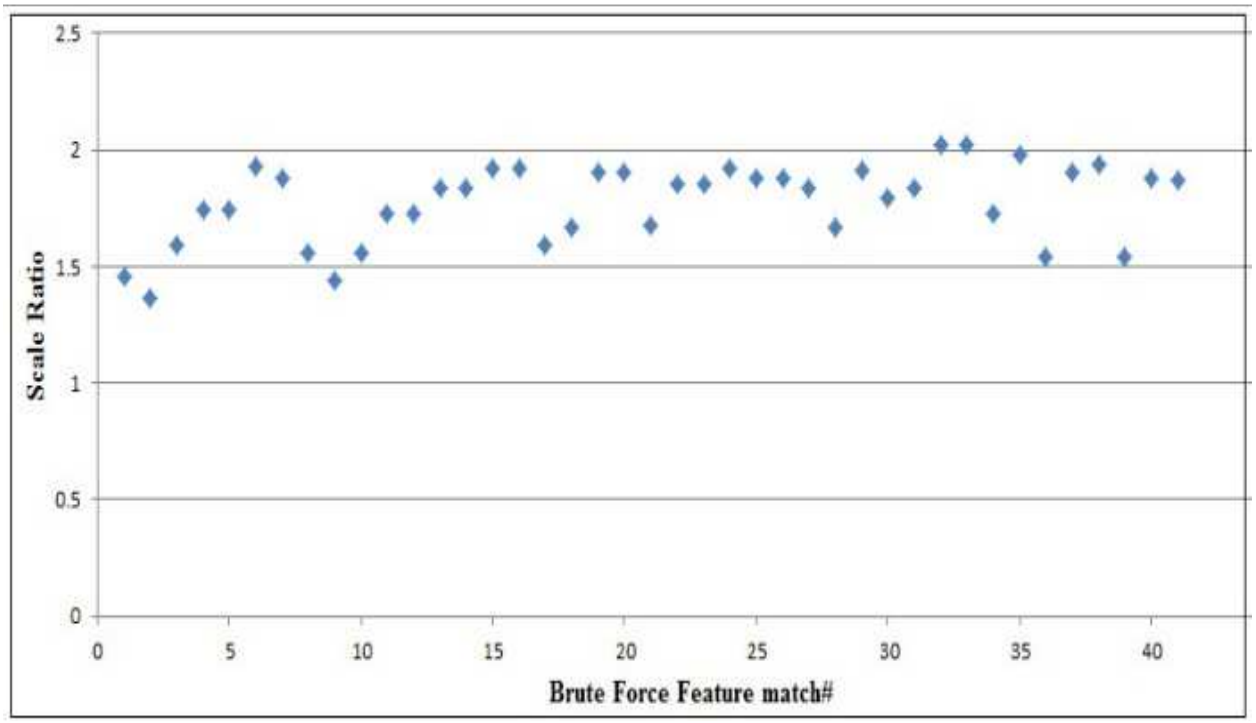

Figure 9 - Filtered scale ratios after using a scale ratio window

In Figure 10, we show the lines from image1 to image2 that correspond to all 61 feature matches in Figure 8. In Figure 11, we show the lines corresponding to matches after scale windowing. Notice that most of the false positive matches are removed.

Figure 10 - Brute Force matching result [11] [15]

Figure 11 - Brute Force matching result filtered with a scale ratio window [11] [15]

3.2 Orientations

Every SIFT feature is given an orientation. The orientation is prominent gradient direction of each feature. When an image is rotated by a certain angle, the orientation of the feature is increased by this angle. Features matched generally have different orientations. The difference of angles between them is constant and gives us the angle of rotation. After using a matching algorithm to find feature matches, we can use the difference in the orientations to remove the wrong matches and find the correct ones.

Our approach is similar to scale ratio windowing described in the previous section. First we, calculate all the differences between the orientations for each feature match. The differences are between 0-360 degrees. We calculate an angle histogram based on the orientation differences. This angle histogram is used to calculate a cumulative histogram. From this cumulative histogram we can find the best possible range for the angle of rotation or the best orientation difference. We do this by sliding a moving window of fixed size along the cumulative histogram and choosing the window that contains the most feature matches. Specifically we search for an angle with highest count using the formula below:

$$count(angle) = Histo[angle + size/2] - Histo[angle - size/2]$$

Then we remove all the feature matches with angle differences that fall outside this window.

This technique is very effective since the differences are scattered from 0-360. It is easy to find a range for orientation difference that satisfies most of the correct feature matches. Fewer false positives remain after windowing. In our experiments, we found that feature matches are filtered more effectively based on orientation difference than scale ratios. Since scale ratios are very close to each other and orientation differences are more spread out. Orientation difference window also works well when the images being matched are skewed.

This process is illustrated in Figures 12 and 13 showing orientation angle differences before and after windowing with the algorithm described above. Figures 14 and 15 show the corresponding lines for feature matches plotted on input images.

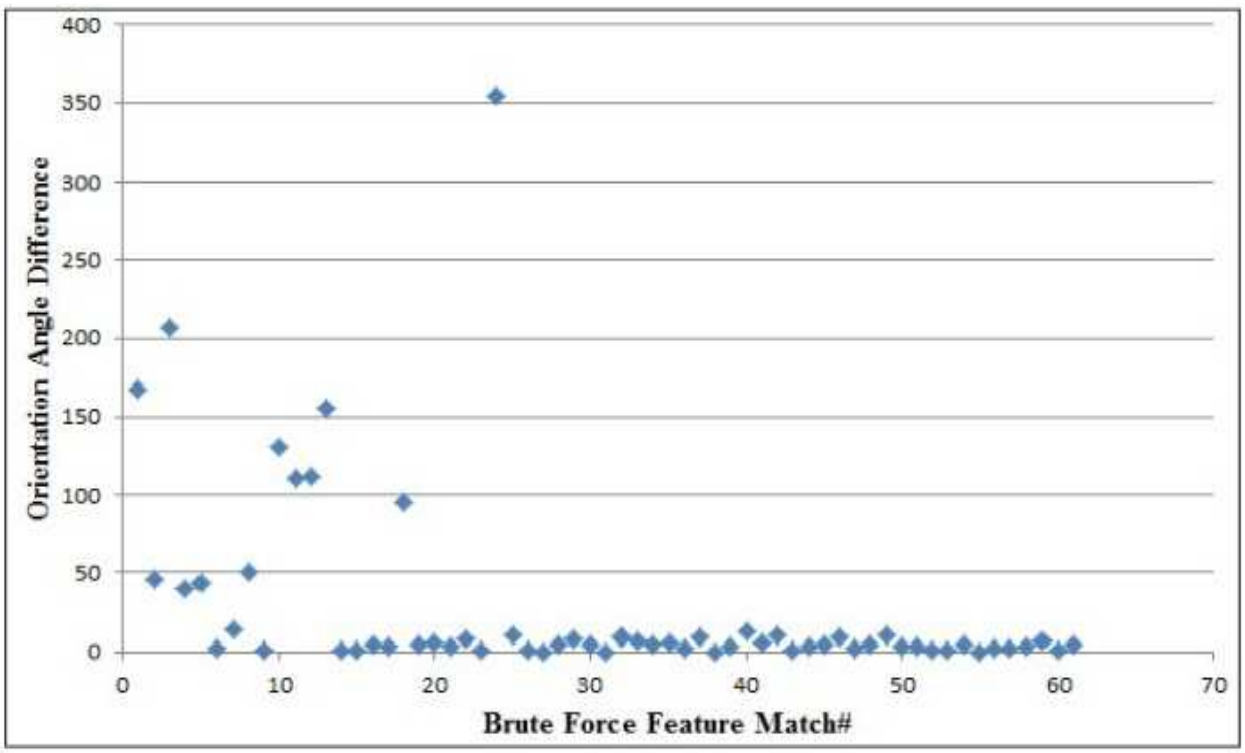

Figure 12 - Orientation angle differences of the matches found using Brute Force Matching Algorithm

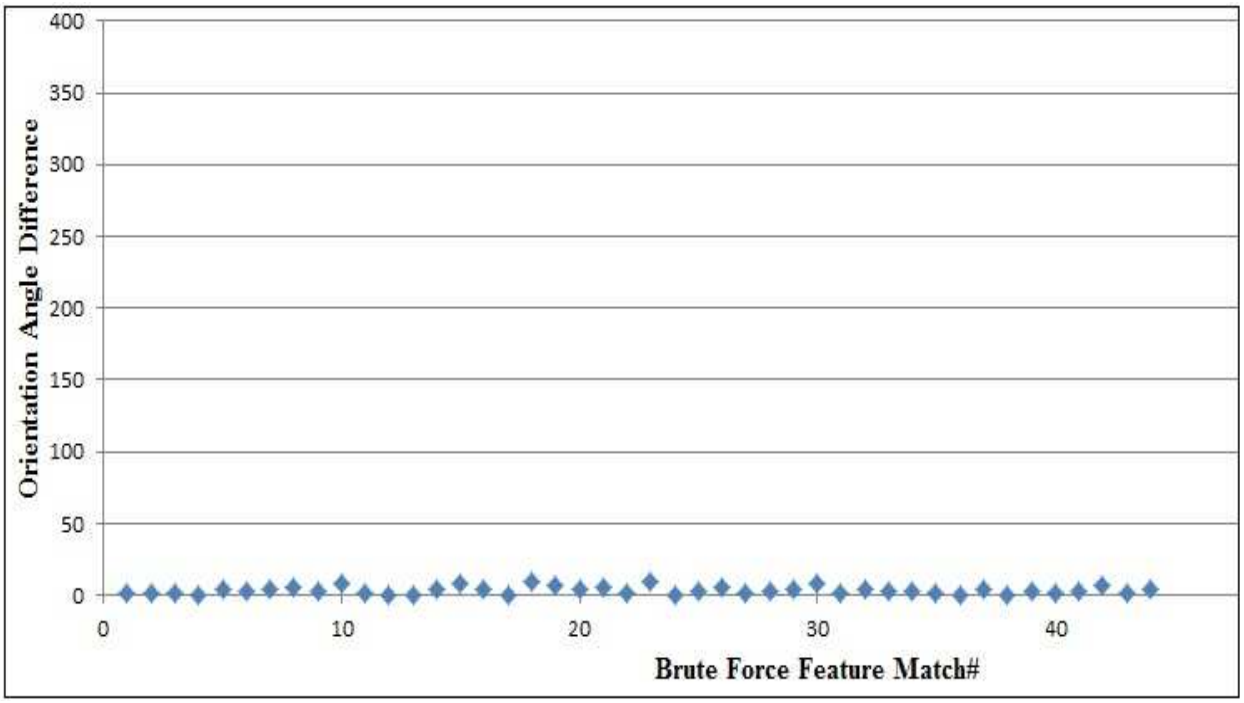

Figure 13 - Filtered orientation angle differences after using orientation angle difference window

Figure 14 - Brute Force matching result [11] [15]

Figure 15 - Brute Force matching result filtered with an orientation angle window [11] [15]

3.3 Creating Counts

The features found in an image by using SIFT technique are intended to be invariant to translations, rotations and scaling transformations and robust to moderate perspective transformations and illumination variations in that image. The number of features found, depend on the image detail and the size of the image. Generally when image size increases the number of SIFT features tend to increase. For a typical 1000x1000 image the number of features found might be in hundreds or thousands. When we want to use these features for further processing like feature matching or object recognition, it can be time consuming. So to decrease the processing time in future it is better to reduce the number of SIFT features by keeping the good features and removing the bad ones.

Our approach is to identify the features that are really scale invariant. To do this we use bilinear interpolation to resize the input image and to see which SIFT feature points remain. First we create N interpolated copies from the given input image. The scale factors used for this interpolation are from 0.5 to 1.5. We do not use the scale factor 1.0 since that would give us the input image as the output. The height and width of the interpolated image is equal to height and width of the input image multiplied by the scale factor.

Next, we calculate SIFT features of these interpolated copies and the input image. Each SIFT feature in an image has three pieces of information. First, the (x, y) location of that feature, second, the Gaussian blur factor of that feature, and finally the vector of 128 feature descriptors.

To see if the SIFT feature is robust to an interpolation we count how many times that feature was found in the interpolated copies. This count tells us if the SIFT feature is robust or not. If the count is equal to N, then the feature is very robust to scale changes. If the count is only 0, 1 or 2, then the feature is not very robust to scale changes. There are two ways to do feature matching between input image and the interpolated images. We can match (x, y) positions, scale and orientation or we can match feature vectors first.

3.3.1 Matching Feature Attributes

For a given feature in the input image find what is the (x, y) position of that feature in the interpolated image. Say the feature was to be found at (x', y') in the interpolated image, if there is a feature at that location then it is fine otherwise search for a closest feature in a certain radius from (x', y'). If we find any feature or features near to (x', y') calculate the scale ratio by dividing the scales of both the features. Then check if this scale ratio is very close to the known scale factor that was used to create the interpolated image. If the scale ratio is approximately equal to the scale factor then we might have a possible match. Since the interpolation does not change the orientation of a feature, calculate the difference between orientations of the features being matched. If the difference is very small we can say that both the features might be equal. Finally compare its descriptor with the corresponding feature descriptors in the input image. Calculate feature difference between each pair by using mean squared error (MSE) between the

128 descriptors values. If the difference is less than a certain threshold we have good feature match and we increment the count for this feature otherwise it is a bad feature match.

3.3.2 Matching Feature Vectors

The other way to do the feature matching is to match feature vectors first. For every feature in the input image we try to find out the best feature match in the interpolated copy by looping over all the features in the input image and the interpolated copy. This way we find the closest feature pair or the best match for each feature in the input image based on MSE difference between 128 descriptors. The pair with least MSE difference would be picked for each feature in the input image. The (x, y) positions are verified by calculating the Euclidean distance between (x, y) at which the feature with the least MSE was found and the location say (x', y') where the feature was expected to be found in the interpolated image. If the distance is less than a certain threshold it is a good match and we increment the count, otherwise it is a bad match. A similar approach can be used to check the scale ratios and the orientation differences of each feature match.

We combine the information that we got from matching the input image and the interpolated copies. We see how many times each feature in the input image got matched or was found in the interpolated copies. Here we consider only the good matches and based on this a count is set for each feature in the input image. Since there were N interpolated copies the count is going to be in between 0…N. This count shows how robust the SIFT feature is to an interpolation. If the count for a feature is high then the chances for that feature to be more scale invariant increase. This kind of behavior helps us find robust features which later can be used in image matching. The hundreds or thousands of features in a big image can be drastically reduced by using this technique, which saves us a lot of time during image matching.

Figure 16 shows number of features per count. Minimum value of count is 0 and the maximum value of count is 10 since we have 10 interpolated copies. Count value is 0 when the feature is not found in any of the interpolated copies. We can say that features with counts from 0-5 are not as robust as features with counts 6-10. Figure 17 shows features plotted for a particular range of counts. If we see the image with features plotted from 6-10, we can say that robust features are less dense and tend to be located in the center of blobs.

Interpolation is used to change X and Y dimensions of an image. The interpolation technique used here is bilinear interpolation. Bilinear interpolation technique can be used to interpolate images up and down to any size. It uses the distance weighted average of adjacent pixels to obtain the output pixel value.

Say if feature1 of input1 image got matched with feature2 of input2 image and the feature vectors for feature1 and feature2 are f1= (a1,a2,a3,a4,a5,a6,a7,…..) and f2= (b1,b2,b3,b4,b5,b6,b7,…..)

Then MSE= $\sum_{i=0}^{i=n}(a_i - b_i)^2$

MSE calculated would always be positive and would show how close these feature vectors are. If MSE value is large (MSE>Threshold) we can say that it was a bad match since the vectors are having a big difference. If MSE is small (MSE<=Threshold) then it is potentially a correct match. We can adjust the threshold to control the number of good, bad matches.

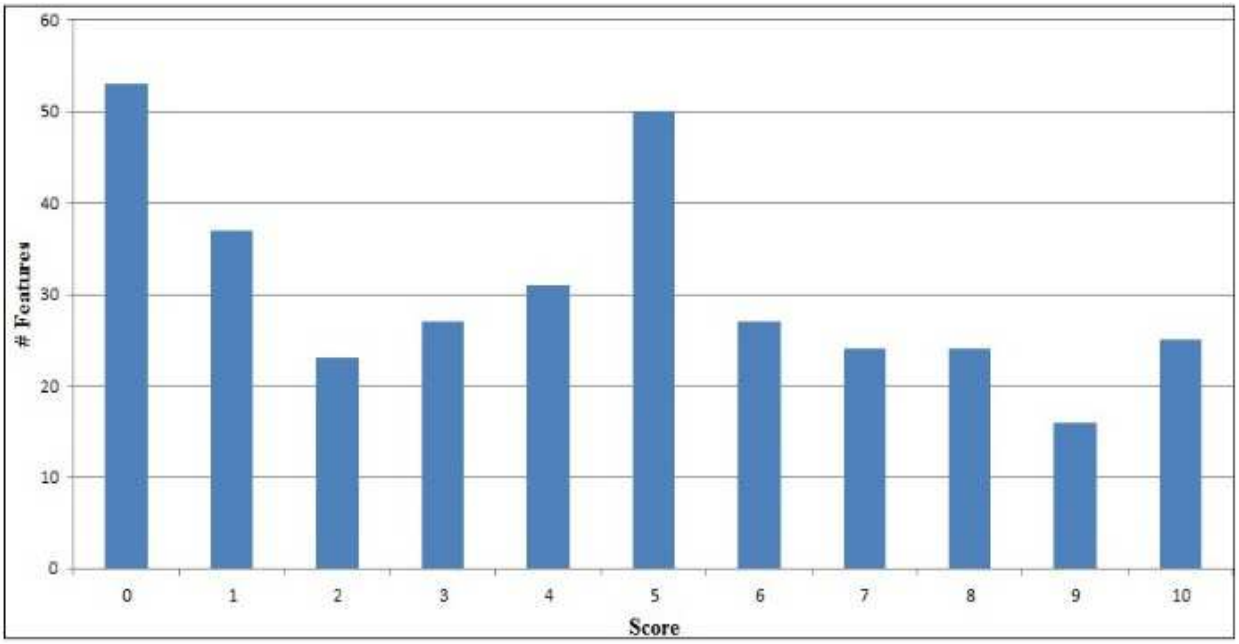

Figure 16 - Number of features per count for the Dr. Pepper logo

Figure 17 - Top-left: SIFT features with counts 0-10, top-right: SIFT features with counts 0-5, bottom-left: SIFT features with counts 6-10, bottom-right: SIFT features with counts 9-10 [11]

3.4 Evaluate Counts for SIFT Feature Matching

In this section, we evaluate the effectiveness of using counts while matching SIFT features. When two images Input1 and Input2 are being matched before the actual matching we need to preprocess both the inputs to find the counts for all features in both the images. The preprocessing might take a few seconds since we create SIFT features for all the interpolated copies of both the images and match them with the original input images. Once the counts are calculated, they are stored in a file so they can be reused.

When Input1 and Input2 are being matched, we only use features with certain counts for matching to see how effective each count is. Features being matched are also selected based on the orientation difference between them. We only consider features with orientation difference less than a certain threshold T. Since in reality we see small amount of rotations we can limit the number of features being matched by their angle of rotation. When T is equal to 360 our method is as same as brute force method for matching. When T is 10 we only match features that got rotated by 10 degrees. When the value of T is low number of feature vectors we try to match are reduced and this saves us time and makes matching faster than the brute force method. The resultant feature matches are further filtered using descriptor difference, orientation window and scale window respectively.

Good SIFT features are selected based on the scores given to each feature. The count for each feature varies from 0 to N (the number of interpolated copies). If we choose a feature that is of higher count it might be a good feature. As we increase count the number of features typically decreases. The features with counts 0-5 are generally bad ones since most of them never get matched. They could be about 30% to 50% of the overall feature set. The features

with counts 6-10 are generally good feature points since they are more scale invariant. Those features are of the 50%-70% of the overall feature set.

Sometimes it is difficult to say which features are good and which ones are bad even though we have a count for each feature. In that case, we can cross check them by trying different images and different count limits to select features and visually inspect them to select a range that works well. To visually inspect the result we add up both the images that are being matched and draw the lines between the features that got matched from Input1 to Input2. We can produce different results based on different MSE Threshold and different count range to see which combination or range is better. The range may slightly vary with different inputs. The results of our SIFT feature matches experiments are described in more detail in chapter 5.

3.5 Evaluate Counts for SURF Feature Matching

SURF (Speeded up Robust Features) is almost same as SIFT. SURF uses determinant of Hessian matrix to locate key points in Gaussian scale space. SURF features have a scale, orientation and descriptors. The feature matching techniques used for SIFT work with SURF as well. SURF is faster than SIFT and generally produces fewer key points when compared to SIFT.

When features in an image input1 are being matched to features in an image input2, there can be two scenarios. We might be looking for all the features in input1 to match some or all of the features in input2. This might be the case in logo matching where input1 is a logo and we are looking for input1 in input2. Next scenario would be where we are expecting few features in input1 to match some or all features in input2. An example for this, could image stitching where part of input1 is same as part of input2. When our count approach was tried on both of the scenarios described above. The use of counts made sense only in the logo matching where input1

is a subset of input2. In this case, when features with higher counts are chosen for matching they perform well. If features with lower counts are chosen they did not perform well.

The use of SURF counts was not as effective for the image stitching scenario. This may be because there were fewer SURF points to begin with, and reducing this number makes it difficult to match images for stitching.

In conclusion, using counts with SURF is much faster than using counts with SIFT (because SURF is so much faster than SIFT) but the benefit of using counts is not as significant as we found for SIFT.

3.6 Scale Test

In the previous section we described how applying a scale window after matching reduces the number of bad matches. In this section, we describe how this scale test can be combined with the matching process to reduce the amount of work further. While matching Input1 to Input2, say Input1 has f1 features and Input2 has f2 features. We have to calculate all possible feature descriptor differences which is equal to f1*f2. The descriptor difference calculations are generally the deciding factor for the speed of any matching algorithm. To reduce the difference calculations, we apply a scale test to all the feature pairs before we calculate the feature differences.

When we compare the scales or blur factor of the all the features pairs that have feature difference less than a certain threshold from Input1 to Input2, we found that the scale comparisons clearly show whether the scale ratio of the correct matches is greater than 1 or less than 1. To make use of this observation, we loop through the features in Input1 and Input2 with feature difference <Threshold and count how many times scale1 of Input1 was less than scale2 of Input2, and scale1 of Input1 was greater than scale2 of Input2. If the range of scale ratios of

the correct matches is not split evenly, then our counts above clearly indicate whether scale ratio of true matches is >1 or <1.

Based on the above concept we created a scale test. In our algorithm first we calculate few feature differences to decide if the scale ratio is greater or less than 1. In our experiments, we found that we got better scale ratio estimate by considering only points in Input1 with scale values 3-6.

Generally when SIFT features are calculated for any image the features with scales 0, 1, 2 are more and features with scales 3, 4, 5, 6 are less. Even though features are distributed unequally among the scales, the probability of matching for features with scales 0, 1, 2 is almost equal to the probability of features with scales 3, 4, 5, 6. This is taken to advantage in our scale test.

After we have tried few combinations for feature differences in the first part of our algorithm we now can say if the scale ratios of the true matches are greater or lesser than one. When we cannot decide whether scale ratios are greater or less than 1 and if the scale ratio range is split over 1 the algorithm defaults to brute force method. Brute force is also used when the total matches are less than 10 since we cannot trust a small number of matches to estimate the scale ratio.

After we know whether the scale ratios of the true matches are >1 or <1, we then apply this scale test to rest of the feature combinations used to calculate the feature differences. Feature differences are greatly reduced after the scale test.

Even though we decided whether the scale ratios of true matches are >1 or <1 still there is a chance of slight error since true matches may have scale ratios of format 1+error or 1-error

and error is very less. In our research we assumed the error to be 0.25 in order to decrease the loss of true matches.

After we have used the scale test we have seen the descriptor calculations get reduced roughly by 35%-50%. This speeds up the matching with a very few matches lost in some cases. The figures below show the results of matching algorithm before and after using the scale test. From Figure 19**,** 20 we see that descriptor differences calculated between inputs with feature match counts 0-10 and 0-10 are reduced from about 900000 to 650000 roughly. From Figures 21, 22 we say that the total matches are preserved in most of the cases except in 4 of the matches. The percentage of correct matches is almost same for both the cases. In this particular example the scale test switches to brute force method or fails to decide the trend of scale ratios when features with counts 0-5 are used from both the inputs. These features with smaller counts are non-robust features so they generate fewer total matches and this is the reason why the scale test fails. We did not include the scale test in our final experiment of finding robust SIFT features because we did not want to risk losing few correct matches.

Figure 18 - Input1 (left) and Input2 (right) used for the scale test experiments in Figure 19, 20, 21, 22 [16]

Figure 19 shows the total number of descriptor comparison operations using invariant counts, angle window and scale window for different ranges of counts for the input image1 and input image2 shown in Figure 18.

Figure 20 shows the reduction in the total number of descriptor comparisons when we integrate the scale test into the comparison loop. Notice the reduction from 900000 maximum in Figure 19 to only 650000 maximum in Figure 20, which is significant.

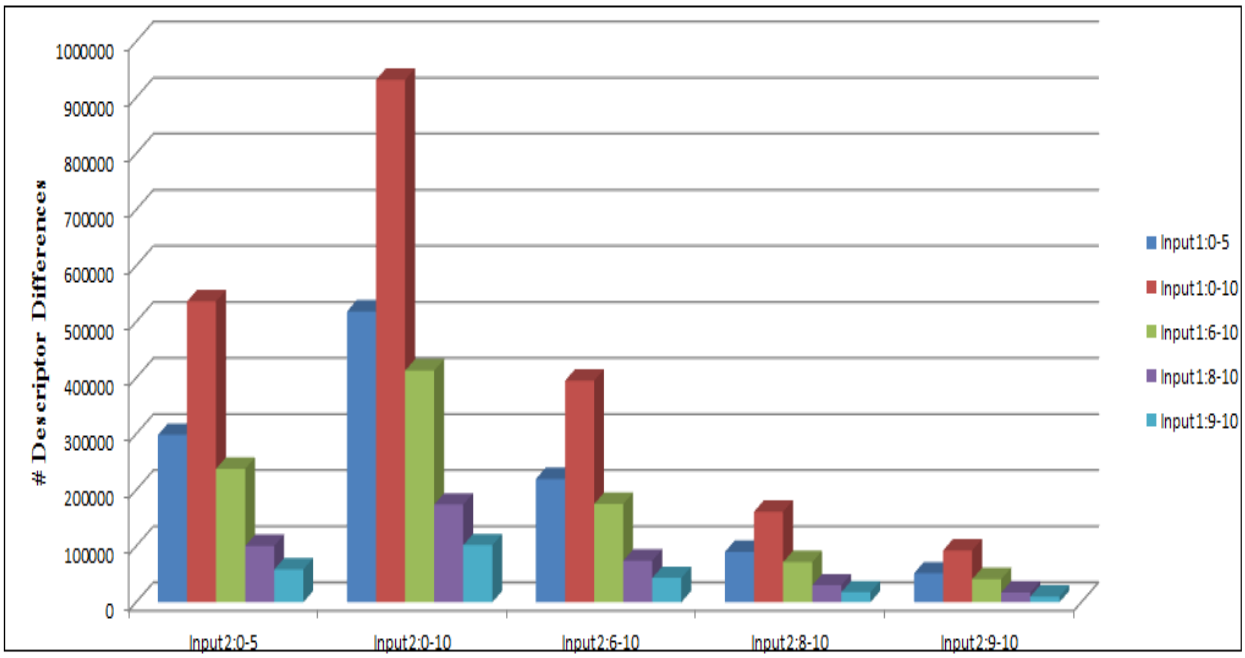

Figure 19 - Number of descriptor differences calculated while matching Input1 and Input2. Matching is done using features with different counts from each input. Scale test was not applied in this case

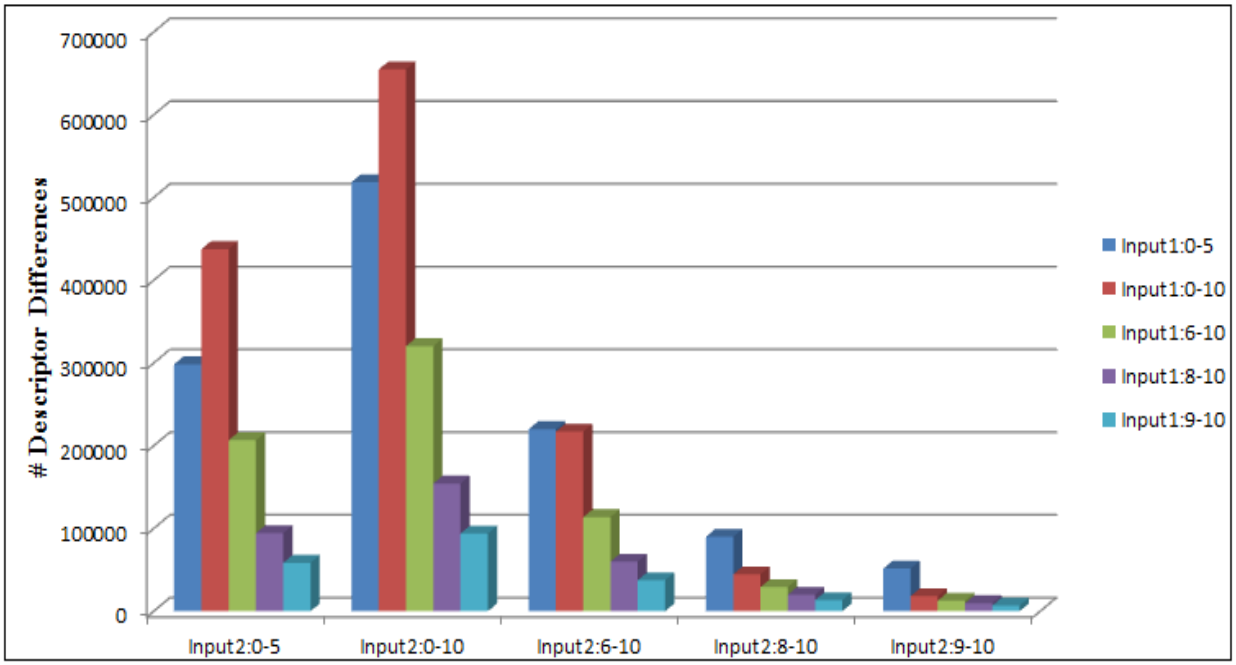

Figure 20 - Number of descriptor differences calculated while matching Input1 and Input2. Matching is done using features with different counts from each input. Scale test was applied in this case

Figures 21 and 22 show the total number of matching features using methods as same as Figures 19 and 20. Notice that these figures are almost identical, so very few true matches were lost using the scale test approach.

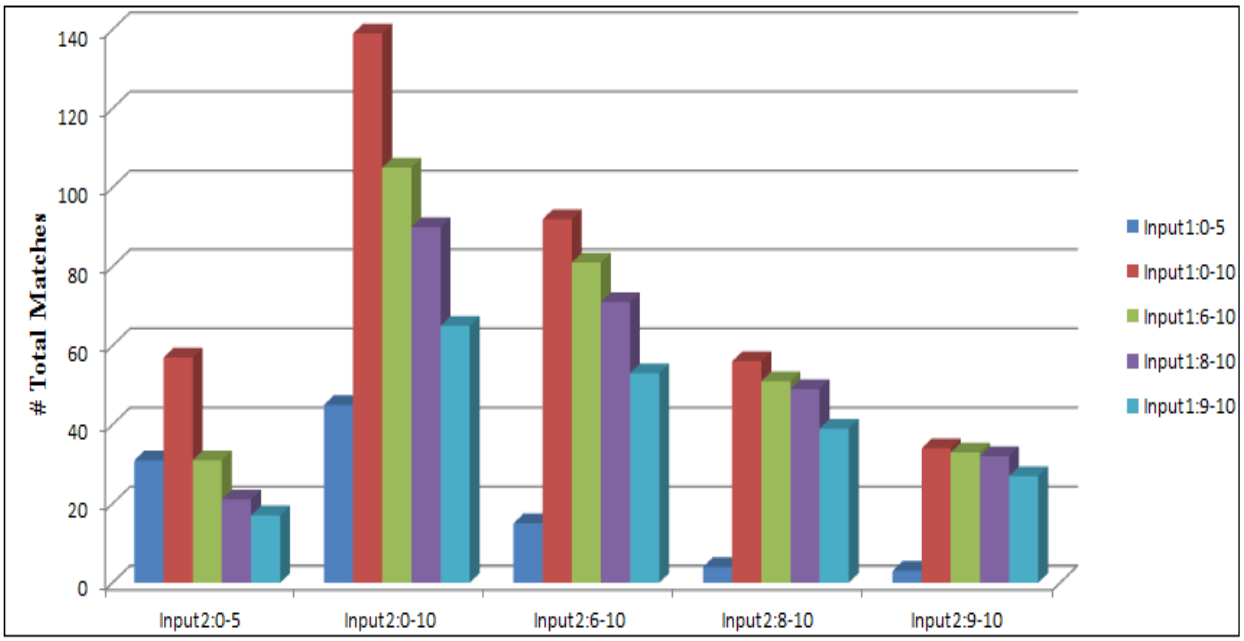

Figure 21 - Total number of matches calculated while matching Input1 and Input2. Matching is done using features with different counts from each input. Scale test was not applied in this case

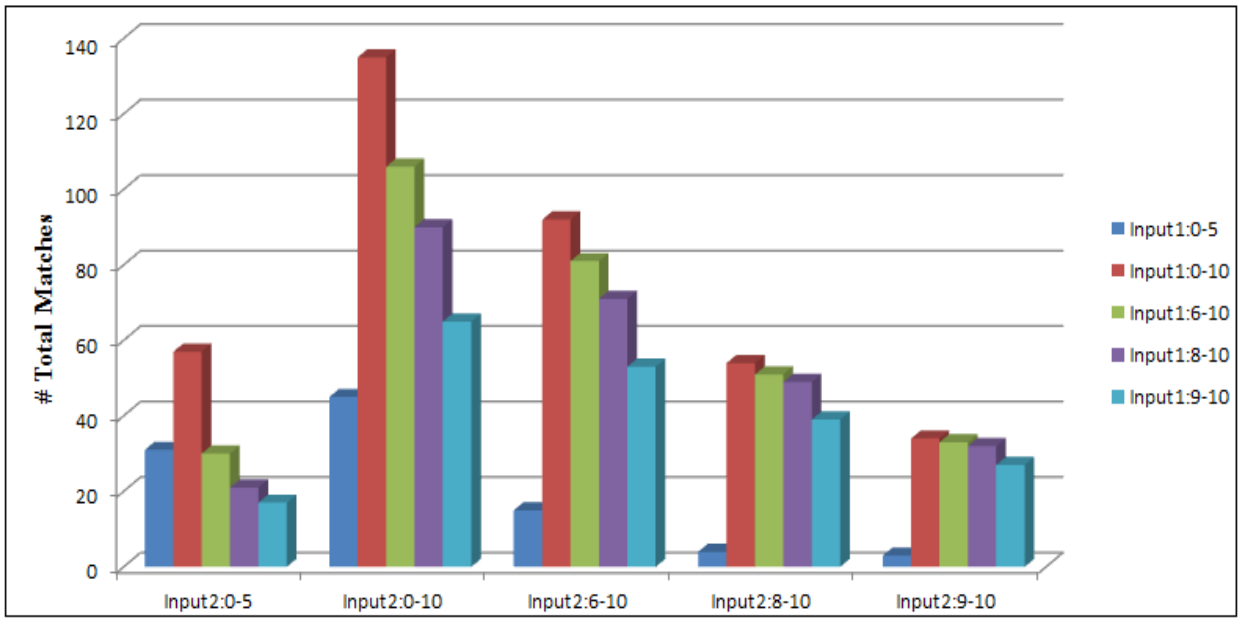

Figure 22 - Total number of matches calculated while matching Input1 and Input2. Matching is done using features with different counts from each input. Scale test was applied in this case

4. TESTING

In this section, we present our testing methods. In section 4.1, we discuss our testing environment. In section 4.2, we present how we measure the performance of feature matching with different invariant counts.

4.1 Testing Environment

SIFT was created and patented by David Lowe. There are other open source versions of SIFT in libraries like OpenCV and VLFeat. These open source versions can be used for non-commercial purposes. We saw that VLFeat SIFT is well documented and is easy to understand so we used it in our research. The majority of key points calculated using VLFeat SIFT correspond almost exactly with Lowes SIFT [7].

Figure 23 - VLFeat feature points (blue) and Lowe SIFT feature points (red) [7]

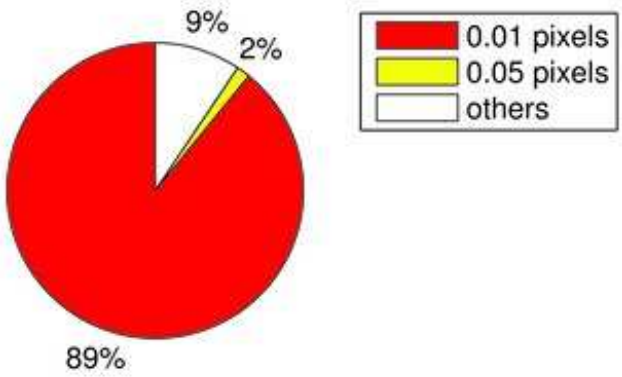

Figure 24 - Percentage of features obtained from VLFeat and Lowe's SIFT that match up to 0.01 and 0.05 pixels [7]

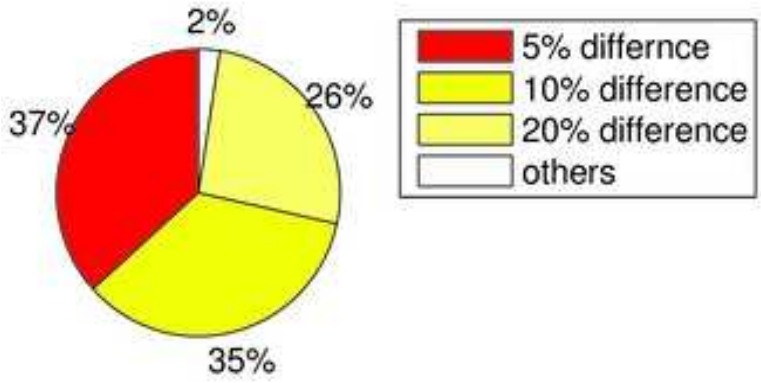

Figure 25 - Percentage of descriptors obtained from VLFeat and Lowe's SIFT whose feature distance is 5%, 10% and 20% of the average descriptor distance [7]

Figures 23 and 24 show that the VLFeat feature points are nearly identical to Lowe's SIFT points. Figure 25 above shows the percentage of descriptors computed by the two implementations whose feature distance is less than 5%, 10% and 20% of the average descriptor distance. This gives us confidence that VLFeat is an accurate implementation.

We did not use the inbuilt matching algorithm in the VLFeat library instead we used the brute force method. VLFeat SIFT was only used to create the SIFT features. The library takes in

an image in PGM format and gives an ASCII file that contains feature location, scale and 128 descriptors. This ASCII file was used for plotting features and matching in our research.

For SURF feature matching we used open source computer vision library (OpenCV). In all our experiments we used OpenCV to calculate the homography matrix that describes the alignment of input images [9].

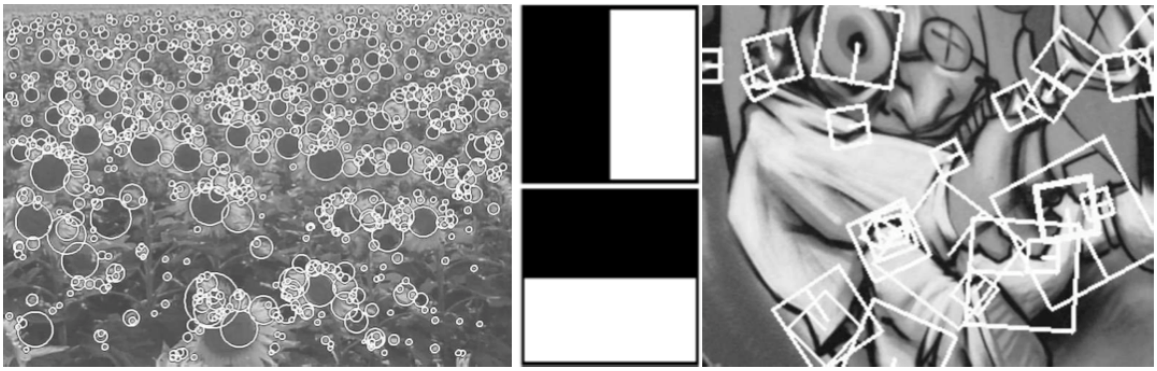

Figure 26 - Example of SURF feature points [4]

4.2 Performance Measurement

The results were generated by matching two input images using different subsets of SIFT feature points. In our system, the minimum count value is 0 and the maximum count value is 10, so we considered feature points in fine count ranges: 0-5, 0-10, 6-10, 8-10, 9-10. We generated 25 results by matching features in the input image1 in one of these ranges to features in input image2 in another range. All the results were generated using an angle of rotation threshold equal to 100.

Graphs were generated to show the total matches, percentage of correct matches, and matching time. The MSE threshold used for the feature difference was 40,000. The percentage of correct matches is measured by calculating the homography that used RANSAC. Time is

measured in the terms of number of descriptor differences calculated during the feature matching.

The 3D-graphs plotted had count ranges 0-5, 0-10, 6-10, 8-10, 9-10 on X and Y axes for input2, input1 respectively. The Z axes shows the value of percentage of correct matches or total matches or number of descriptors differences calculated for each combination of count ranges on X and Y axes.

The idea was to see how well each count range performed and how many features can be kept or removed based on their counts without effecting the matching. We generated an image for each case to see how the alignment of the bounding rectangle changes for different combination of count range.

Once the matching was done for each case, using the correct matches we found the projected scale and orientation windows. These windows are very accurate in most of the cases. If our program was being used to process a video these projected scale and angle of rotation windows can be taken into consideration before processing the next frame.

5. RESULTS

In this section, we present results of the experiments outlined in the previous section. We tested feature matching with a variety of test images, using a variety of scale invariant counts. We measured performance based on three metrics: total number of matches, percentage of correct matches, and number of descriptor operations done during the matching.

We performed our analysis with 30-35 pairs of images and found a high degree of uniformity between the test results from each image pair. Below is a detailed account of our findings with four image pairs: a Dr. Pepper logo, the U of A union living room, the U of A union fountain and a stop sign.

5.1 Dr. Pepper

Figures 27, 28 and 29 show the results of SIFT feature matching of Dr. Pepper logos with different sets of SIFT features. In Figure 27 we show matches using all SIFT points. In Figure 28 we show matches with only the good points, and finally in Figure 29 we show matching with insufficient number of SIFT points.

Figure 27 - Dr. Pepper logo matched from Input1 (left) to Input2 (right) using features with counts 0-10 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [12] [13]

Figure 28 - Dr. Pepper logo matched from Input1 (left) to Input2 (right) using features with counts 6-10 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [12] [13]

Figure 29 - Dr. Pepper logo matched from Input1 (left) to Input2 (right) using features with counts 9-10 (Input1) and 8-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [12] [13]

Figure 30 shows that Input1 has more features with counts 5-10 when compared to features with counts 0-4. In general, we can say if Input1 was used for matching it would give good results since most of the features are scale invariant. Figure 31 shows that Input2 has features evenly distributed among the counts. There are about 500 features which have a count of zero and 500 is the highest value in the histogram (Figure 31). In general we can say that Input2 would give many errors during any sort of matching since there are fewer robust features with counts 6-10 when compared to the features with counts 0-5.

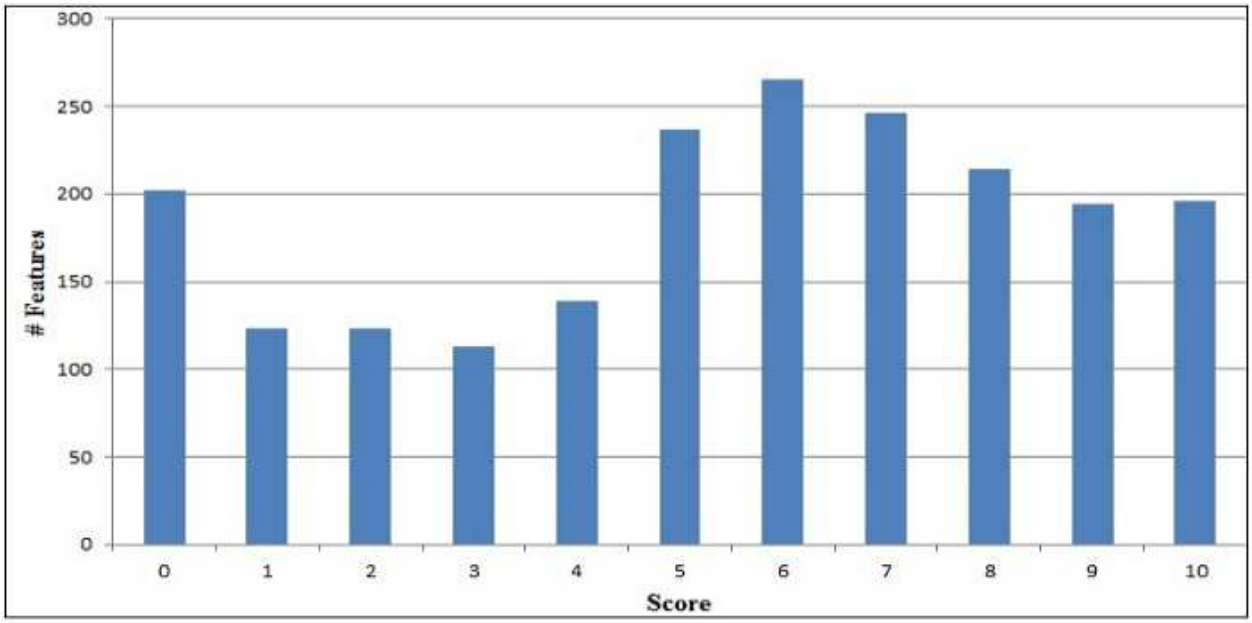

Figure 30 - Number of features per count for Input1

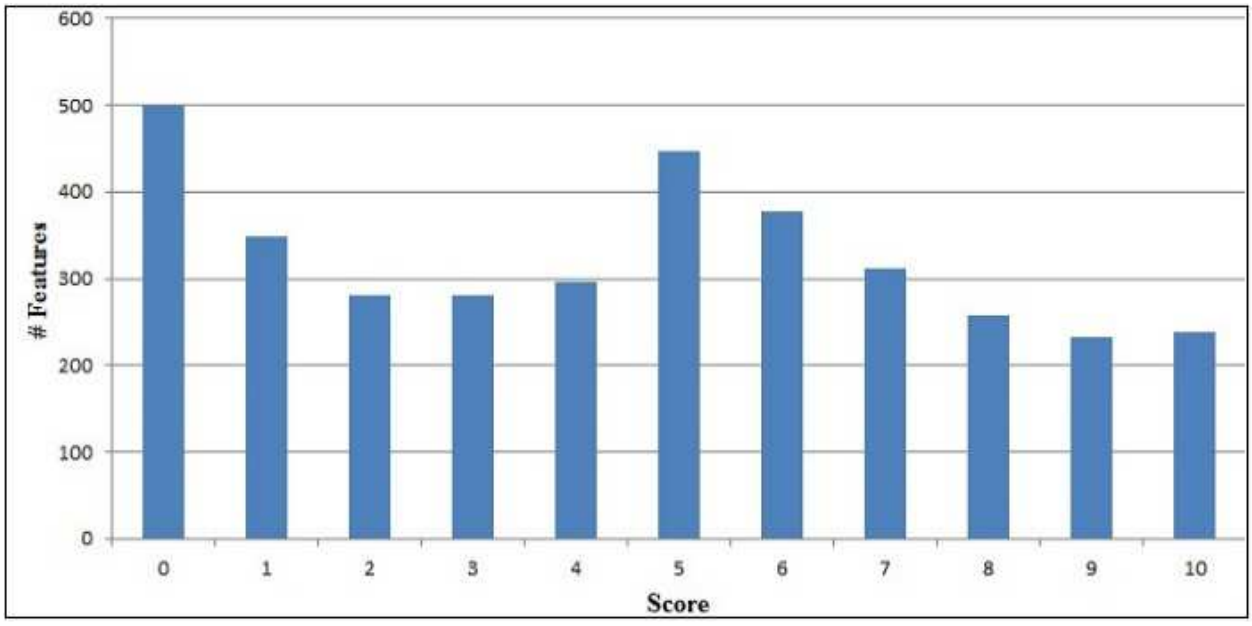

Figure 31 - Number of features per count for Input2

Figures 32, 33 and 34 show the results of all 25 matching experiments. In Figure 32 we have total number of matches. In Figure 33 we have percentage of correct matches, and Figure 34 we have the descriptor differences.

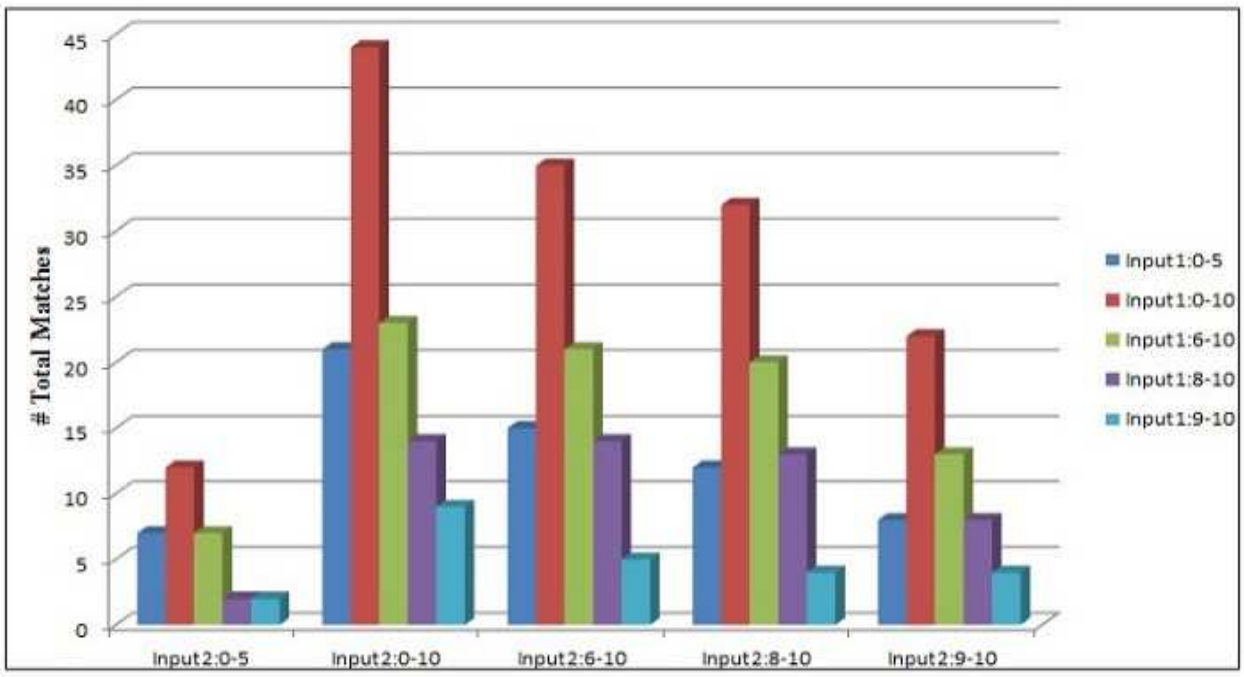

Figure 32 - Total matches between Input1 and Input2. Matching is done using features with different counts from each input

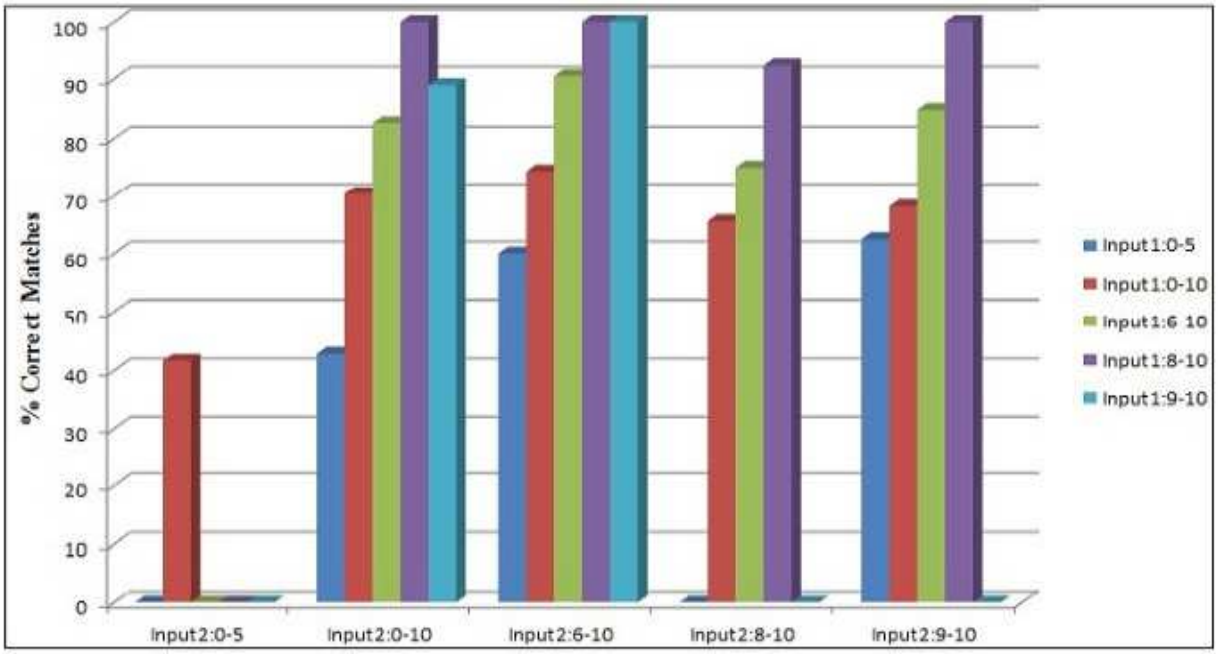

Figure 33 - Percentage of correct matches between Input1 and Input2. Matching is done using features with different counts from each input

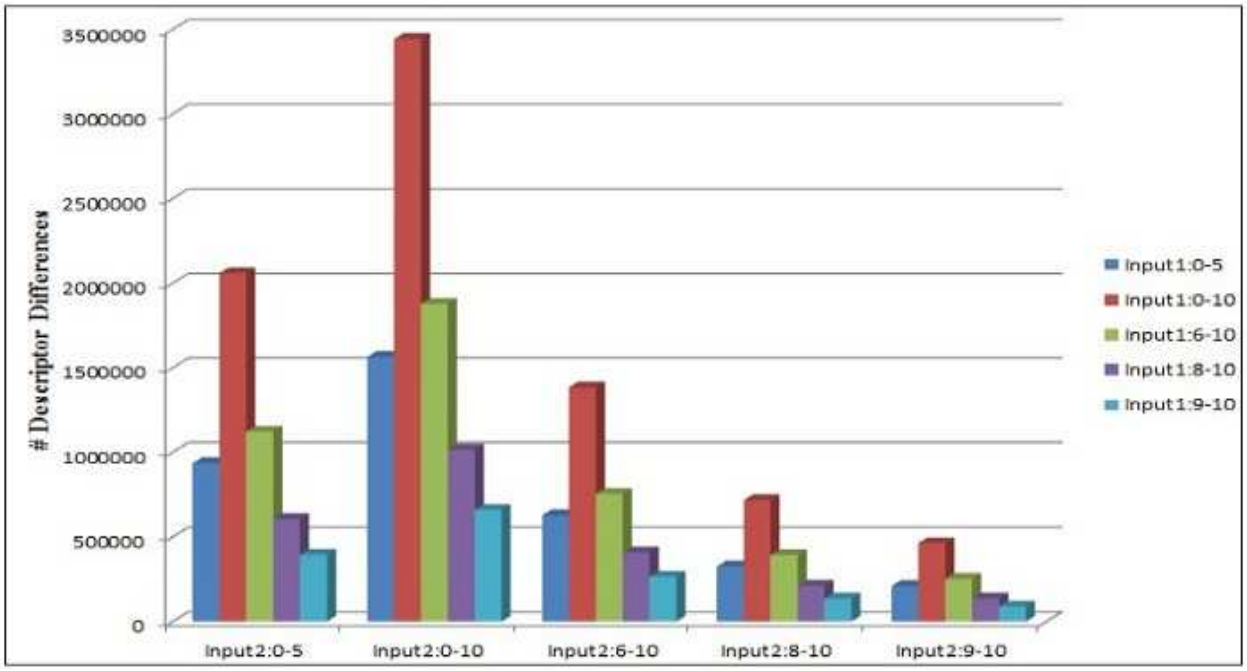

Figure 34 - Number of descriptor differences calculated during the feature matching between Input1 and Input2. Matching is done using features with different counts from each input

Our first result shows matching with count ranges are 0-10(Input1) and 0-10(Input2) (Figure 27). The Total number of matches is equal to 44 and the percentage of correct matches is approximately 70. The percentage of correct matches is less as we expected since non-robust features with low counts are more in Input2. The descriptor differences calculated are 3445038. The differences calculated are certainly very high. The comparison between all the feature vectors is required and cannot be avoided.

The best combination of count ranges for matching in this example is 6-10 and 0-10 (Figure 28). The descriptors differences calculated in this case are 1878986 and that is almost 50% of reduction when compared to the previous case, where we used features of count ranges 0-10 and 0-10. This certainly shows that use of robust features with higher counts for matching saves us some time. Here the percentage of correct matches is 82. Even though we were able to remove most of the wrong matches we also removed few correct ones but the overall efficiency of our matching algorithm was improved.

The worst combination of counts in this example is 9-10 and 8-10 (Figure 28). The total matches in this case are 4, which is bad since there are not enough matches for us to create a homography and verify whether few or all of them are correct. Generally as we consider higher count values, the number of feature points decrease rapidly, so using very high count ranges might not work in some cases.

5.2 Union Living Room

Figures 35, 36 and 37 show the results of SIFT feature matching of union living room with different sets of SIFT features. In Figure 35 we show matches using all SIFT points. In Figure 36 we show matches with only the good points, and finally in Figure 37 we show matching with non-robust SIFT points from input image1.

Figure 35 - Input1 (left) to Input2 (right) are matched using features with counts 0-10 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

Figure 36 - Input1 (left) and Input2 (right) matched using features with counts 6-10 (Input1) and 6-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

Figure 37 - Input1 (left) to Input2 (right) matched using features with counts 0-5 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

Figures 38 and 39 we show the number of features per count for Input1 and Input2. We can say that majority of features in both the inputs are within count range of 6-10. Generally when more features have high counts the number of correct matches and total matches tend to be larger if there is any significant match between inputs.

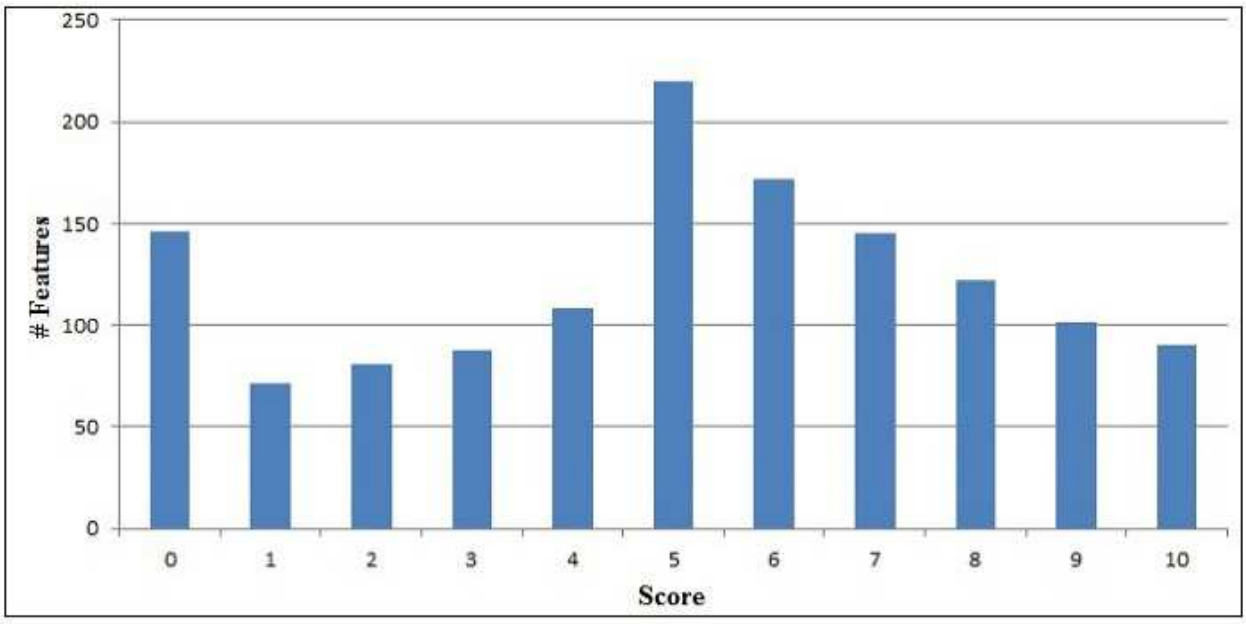

Figure 38 - Number of features per count for Input1

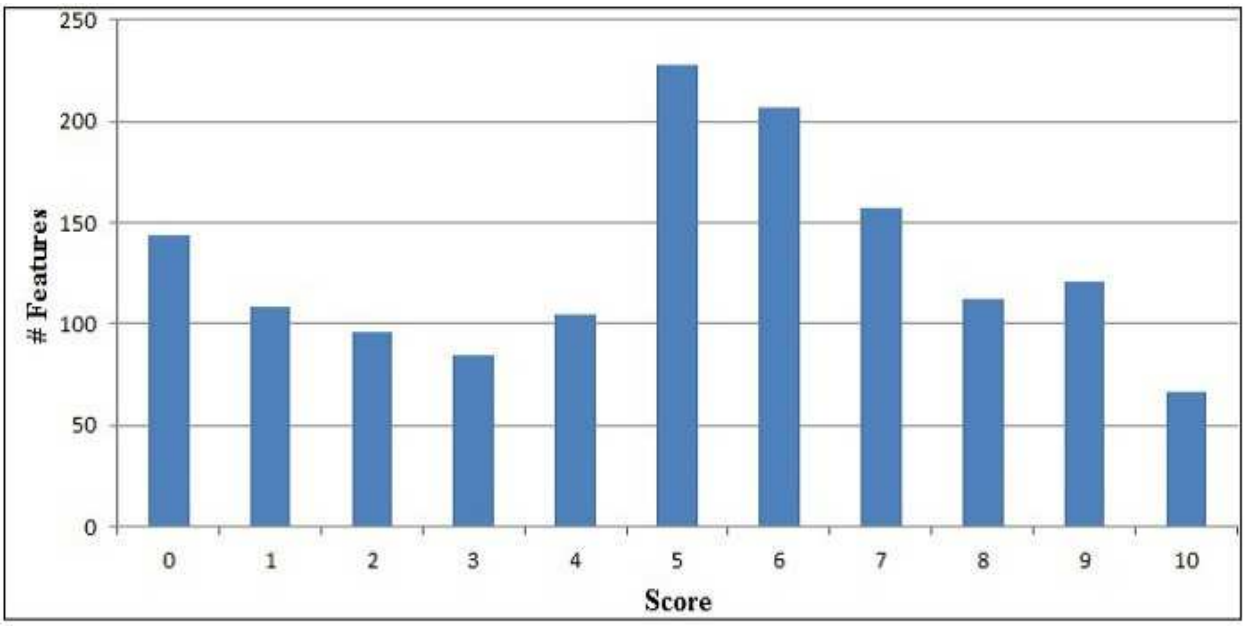

Figure 39 - Number of features per count for Input2

Figures 40, 41 and 42 show the results of all 25 matching experiments. In Figure 40 we have total number of matches. In Figure 41 we have percentage of correct matches, and Figure 42 we have the descriptor differences.

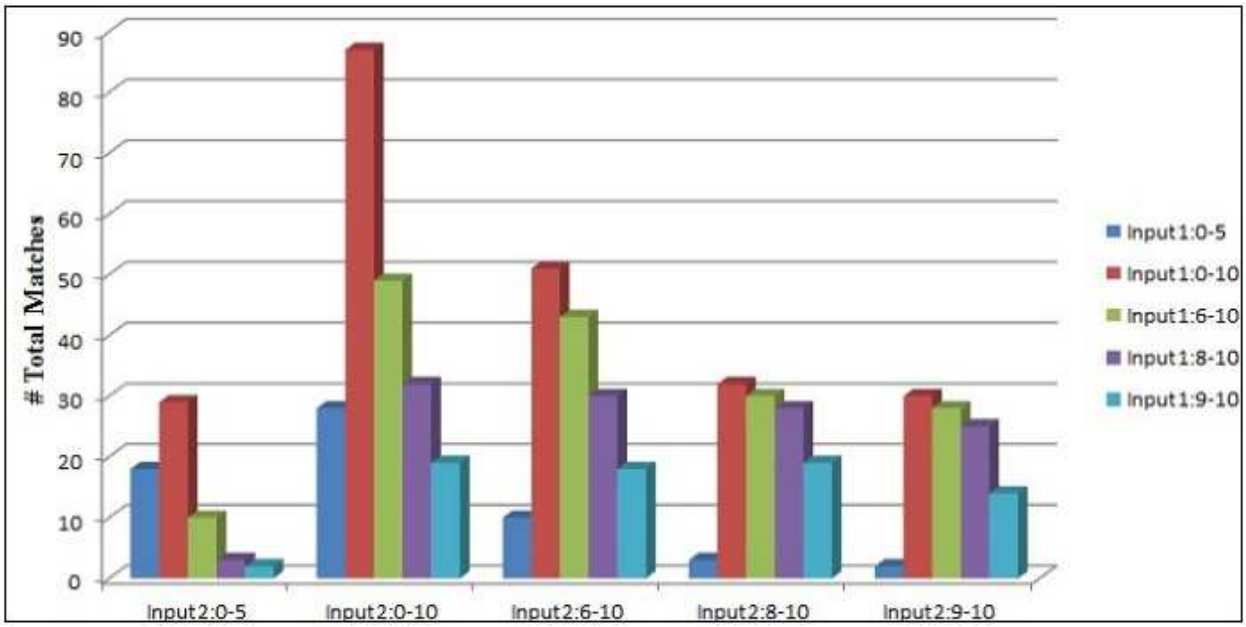

Figure 40 - Total matches between Input1 and Input2. Matching is done using features with different counts from each input

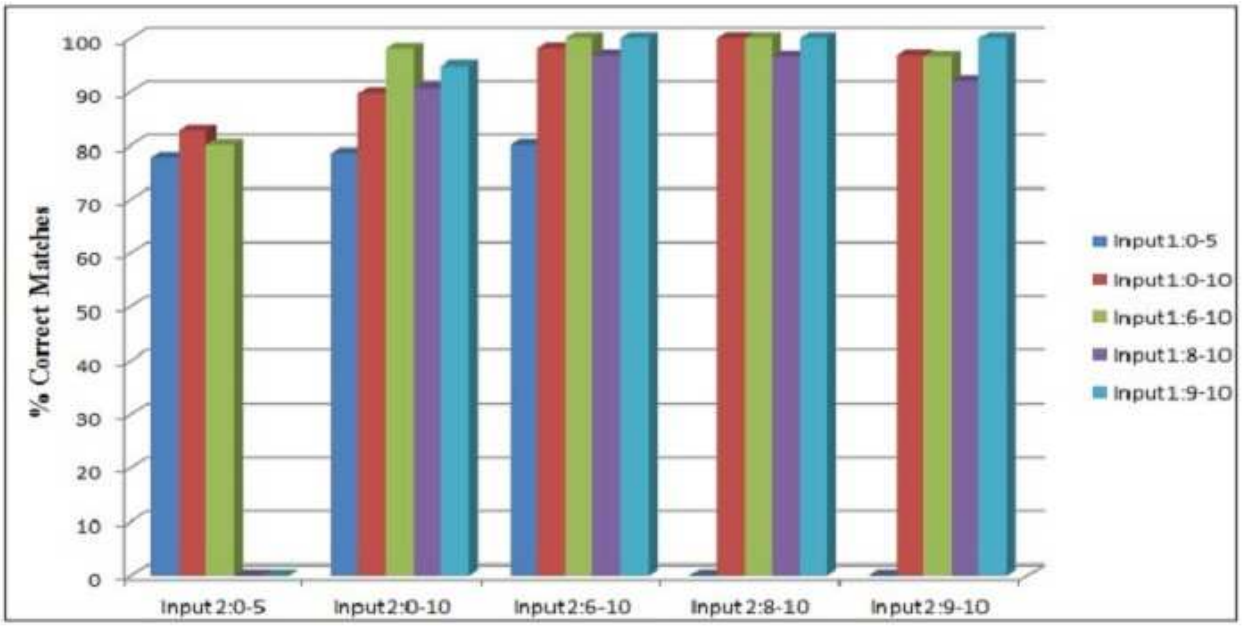

Figure 41 - Percentage of correct matches between Input1 and Input2. Matching is done using features with different counts from each input

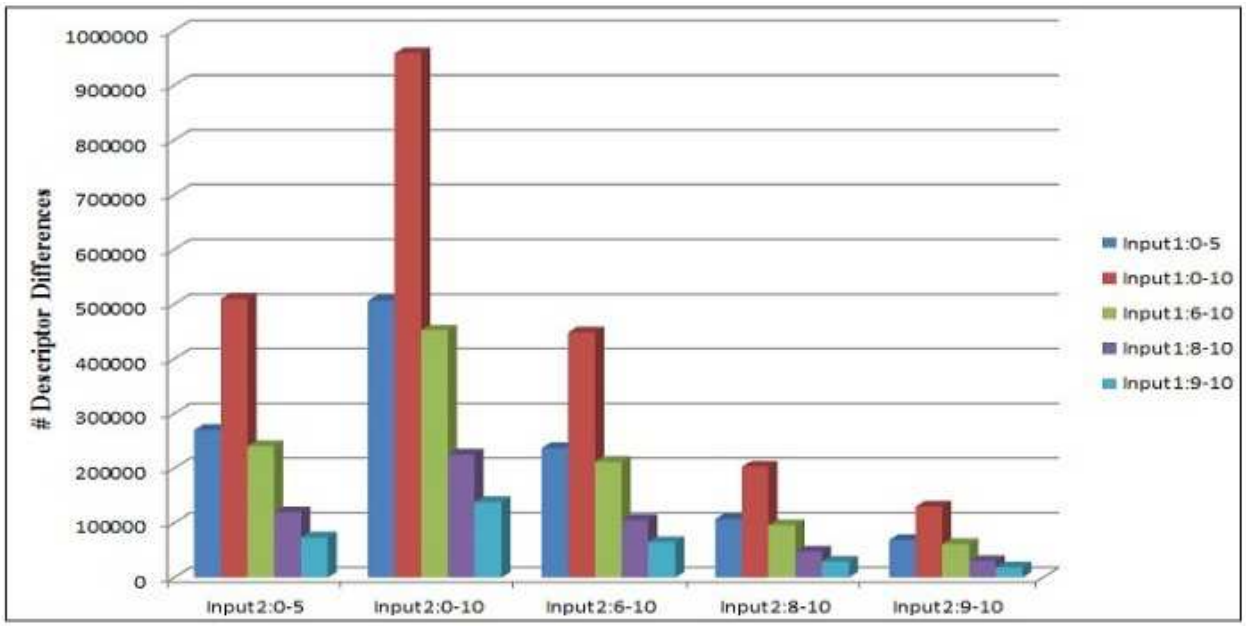

Figure 42 - Number of descriptor differences calculated during the feature matching between Input1 and Input2. Matching is done using features with different counts from each input

The result in which all the features where used had count ranges from 0-10 and 0-10 (Figure 35). The percentage of correct matches in this case was about 90% and the total matches were 87. The percentage of correct matches is very high even when we used all the features for matching.

The best combination of count ranges for matching in this example is 6-10 and 6-10 (Figure 36) since it has 43 correct matches and the percentage of correct matches was 98%. The descriptor differences calculated in the best case are 211715. This is about 75% lower when compared with the differences calculated in the combination 0-10 and 0-10.

The worst case here was 0-5 and 0-10. In this case the total matches are equal to 28, which is very low compared to the previous cases. The percentage of correct matches is 78% but this is also lower than the previous cases. This clearly shows that features with counts 0-5 are not robust.

In this example since there are few errors and high number of total matches, the matching algorithm performs well even in the higher count range.

5.3 Union Fountain

Figures 43, 44 and 45 show the results of SIFT feature matching of union fountain with different sets of SIFT features. In Figure 43 we show matches using all SIFT points. In Figure 44 we show matches with only the good points, and finally in Figure 45 we show matching with non-robust SIFT points from input image1 and input image2.

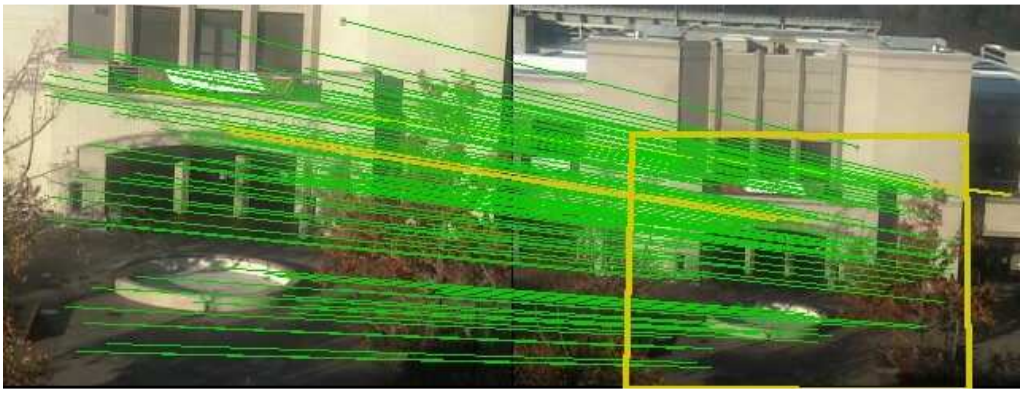

Figure 43 - Input1 (left) and Input2 (right) matched using features with counts 0-10 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

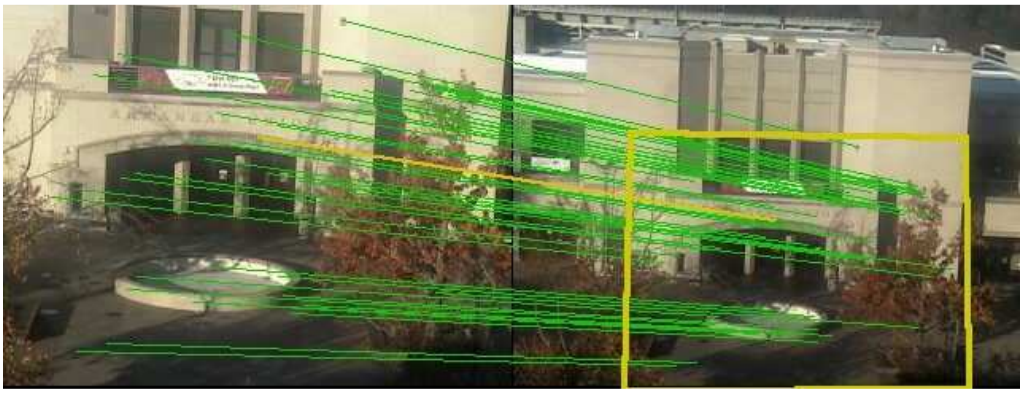

Figure 44 - Input1 (left) and Input2 (right) matched using features with counts 9-10 (Input1) and 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

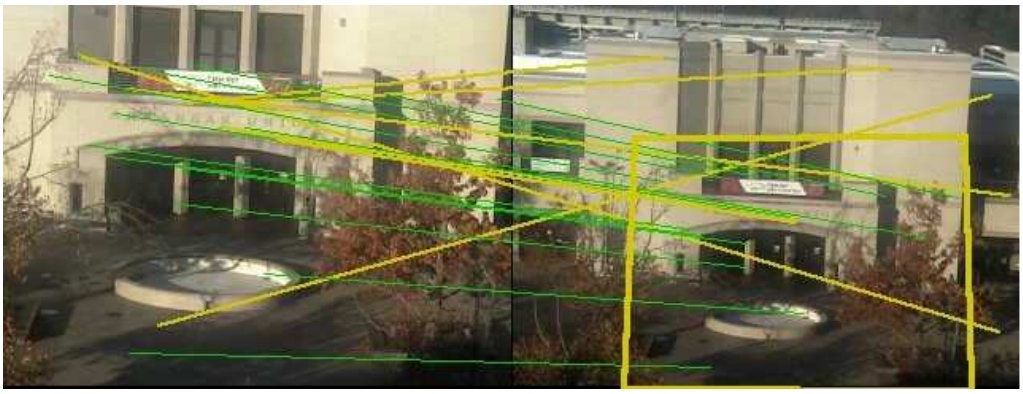

Figure 45 - Input1 (left) and Input2 (right) matched using features with counts 0-5 (Input1) and 0-5 (Input2). Matches drawn in green (correct) and yellow (wrong) [16]

In Figures 46, 47 we show the number of features with counts between 0 and 10. As in the previous experiment, the number of feature points with counts 5-7 is higher than the other count values. This indicates that the SIFT features are scale invariant and will be good for matching.

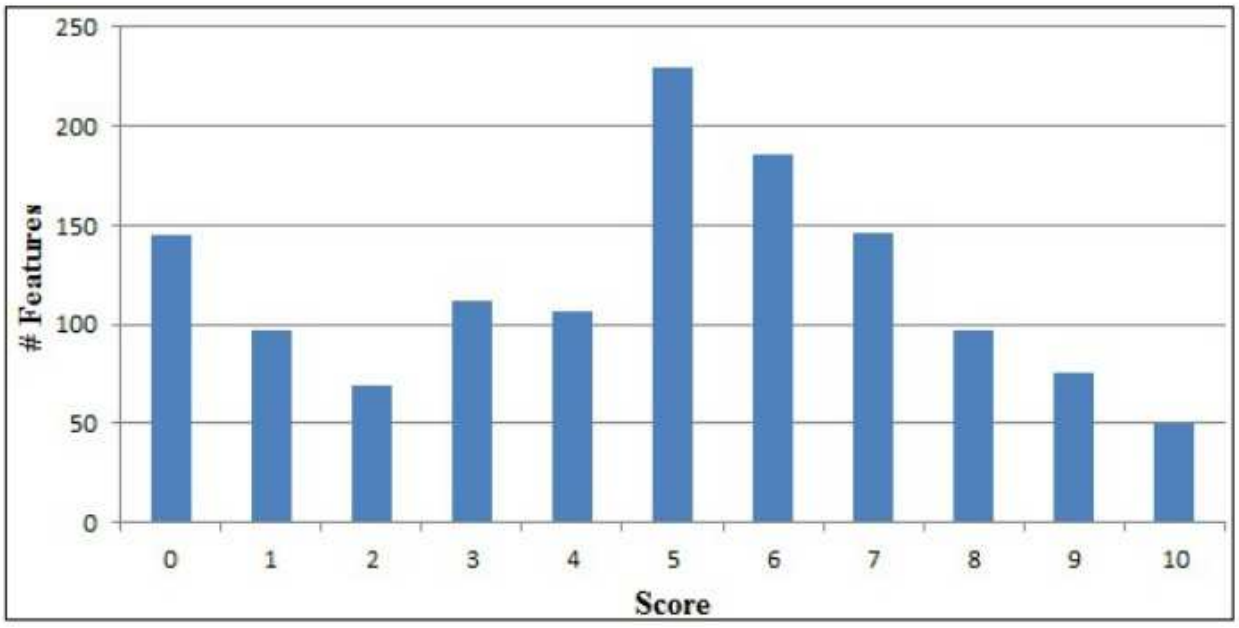

Figure 46 - Number of features per count for Input1

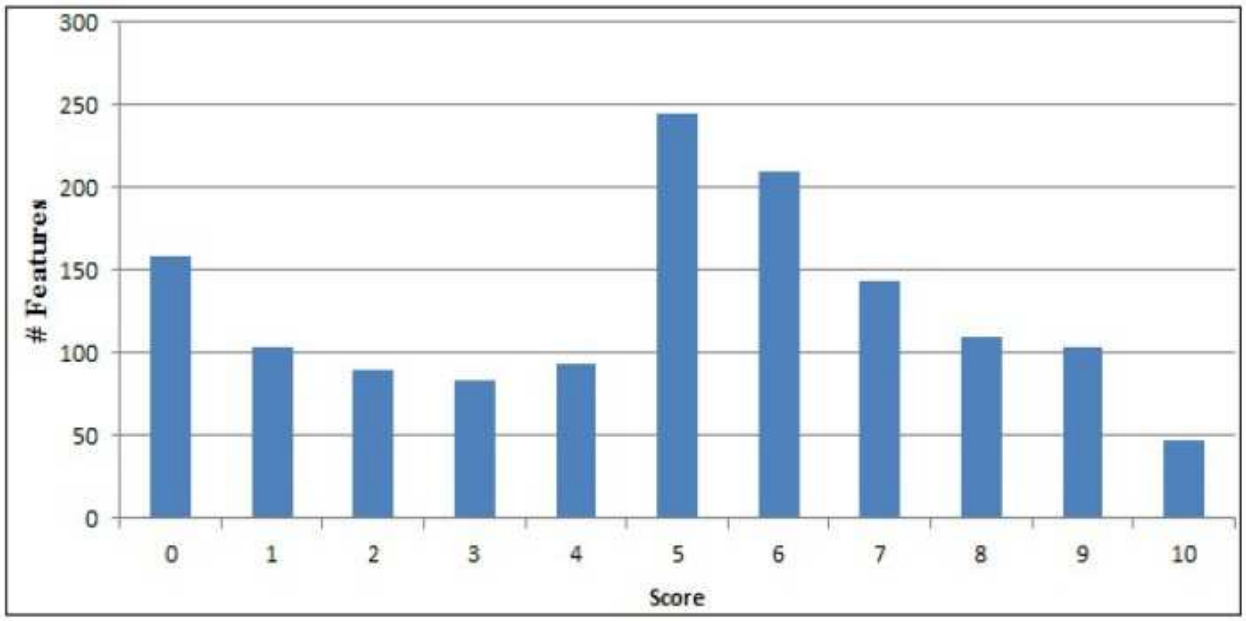

Figure 47 - Number of features per count for Input2

Figures 48, 49 and 50 show the results of all 25 matching experiments. In Figure 48 we have total number of matches. In Figure 49 we have percentage of correct matches, and Figure 50 we have the descriptor differences.

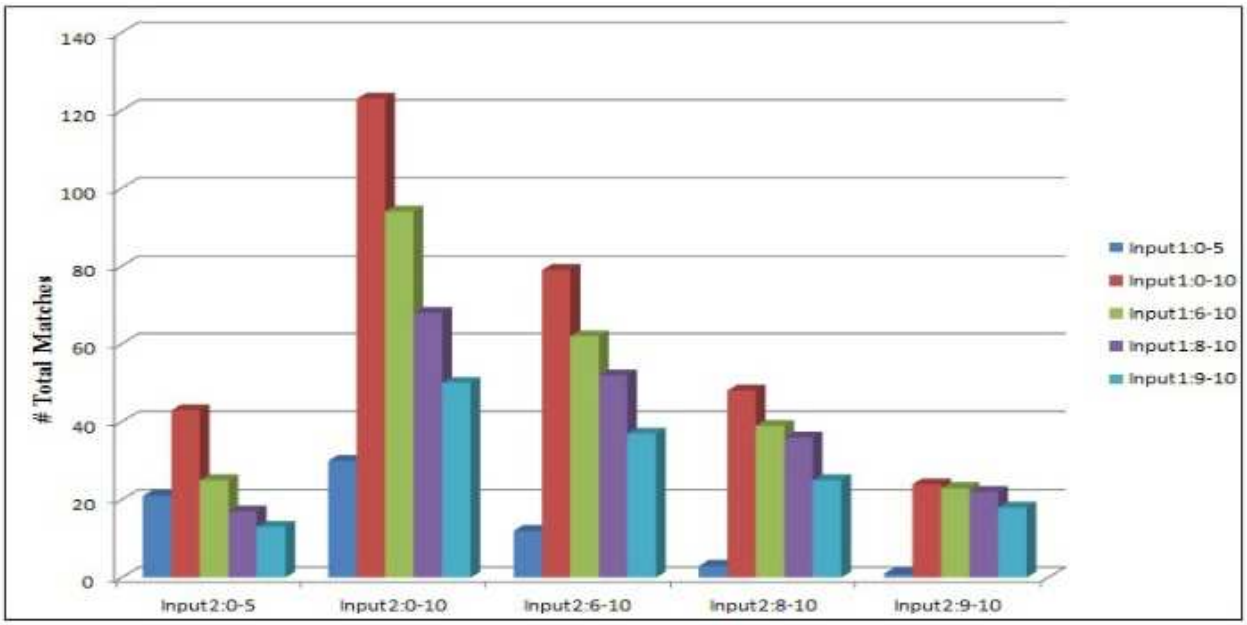

Figure 48 - Total matches between Input1 and Input2. Matching is done using features with different counts from each input

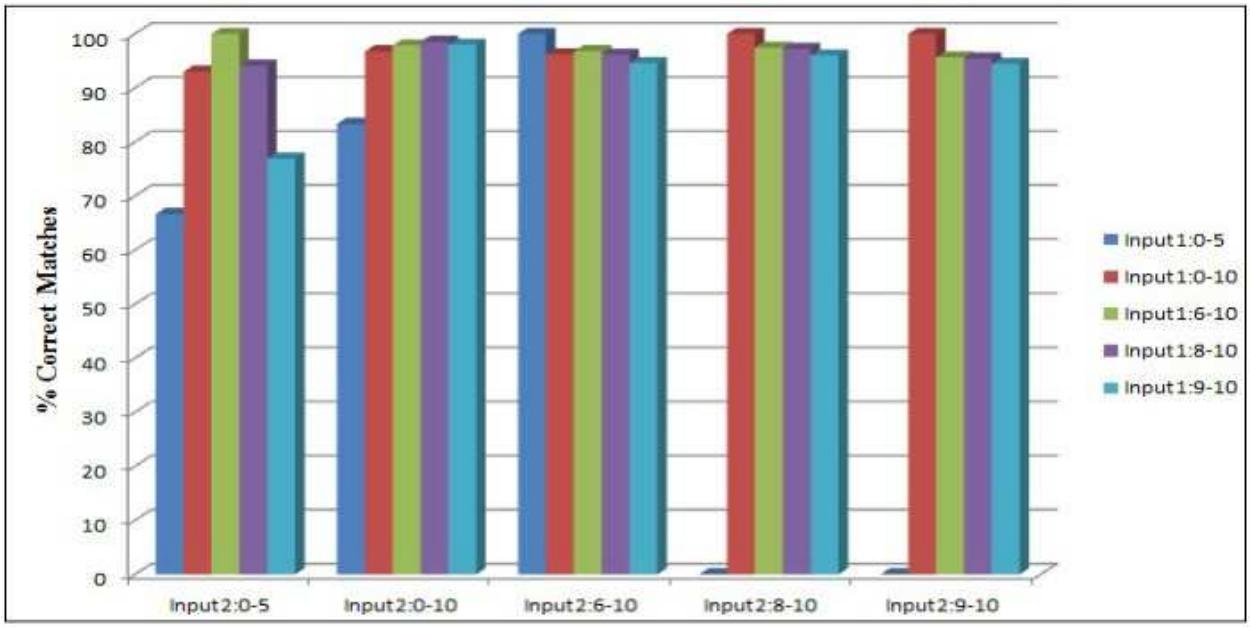

Figure 49 - Percentage of correct matches between Input1 and Input2. Matching is done using features with different counts from each input

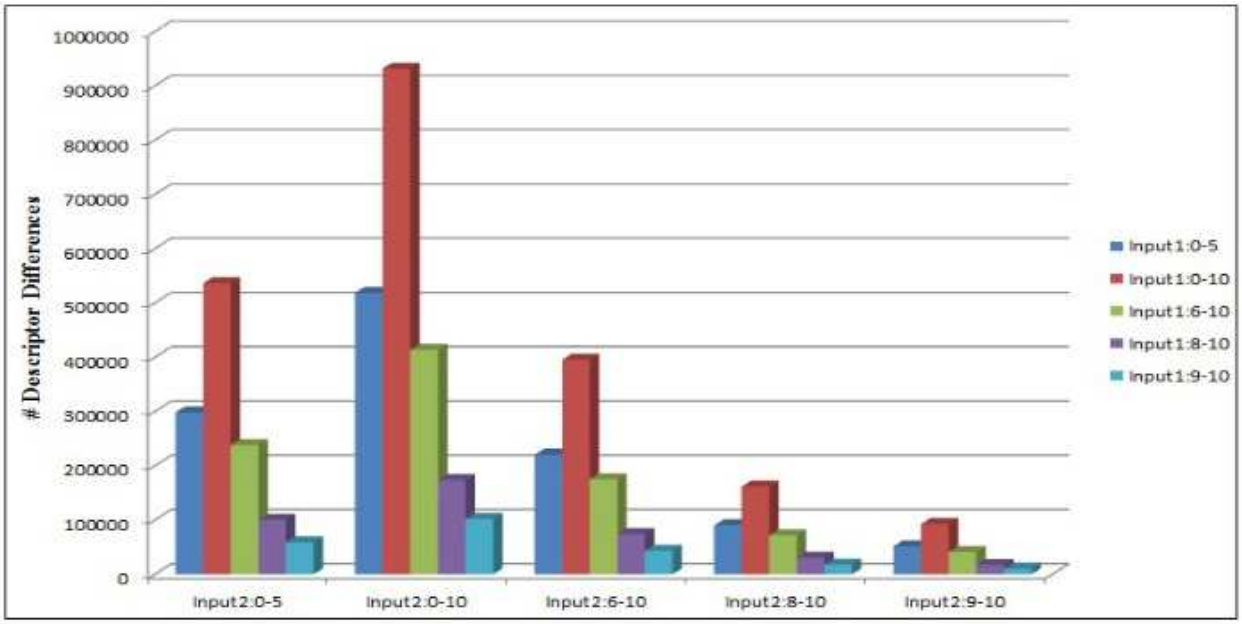

Figure 50 - Number of descriptor differences calculated during the feature matching between Input1 and Input2. Matching is done using features with different counts from each input

The result where all the features where used for the matching, total matches are 123, percentage of correct matches is 96% and, total descriptor differences calculated are 931715 (Figure 43). The percentage of correct matches is very high and very few wrong matches were found.

The best combination of count ranges in this example is 9-10 and 0-10 (Figure 44). The number of total matches is equal to 50, which is less than half of the total matches found in the previous case. This is because we have used a very high count range 9-10 for Input1. Percentage of correct matches in this case is 98%. Even though we used fewer features for our matching, the percentage of correct matches is still very high. Descriptor differences calculated in this case are 101763, which is almost 89 % less than the descriptor differences calculated in the previous case where count ranges used are 0-10, 0-10. This gives us a huge time savings in feature matching.

The worst combination of count ranges in this example is 0-5, 0-5 (Figure 45). Total matches in this case are 21, percentage of correct matches is 66% and the descriptor differences calculated are 298485. The percentage of correct matches is the lowest among all the count range combinations. This proves that features with lower counts are not robust.

The success rate of features with counts 0-5 is less in most of the examples we tried. The matching would be much faster without these features and removing these features will not hurt the efficiency of the matching algorithm.

5.4 Stop Sign

In this section, we show some preliminary results of using invariant counts with SURF features. In general there are fewer SURF features for an image than SIFT features, and they are roughly uniform distribution of count values. Figures 51, 52 and 53 show the results of SURF feature matching of stop sign with different sets of SURF features. In Figure 43 we show matches using all SURF points. In Figure 44 we show matches with only the good points, and finally in Figure 45 we show matching with non-robust SURF points from input image1.

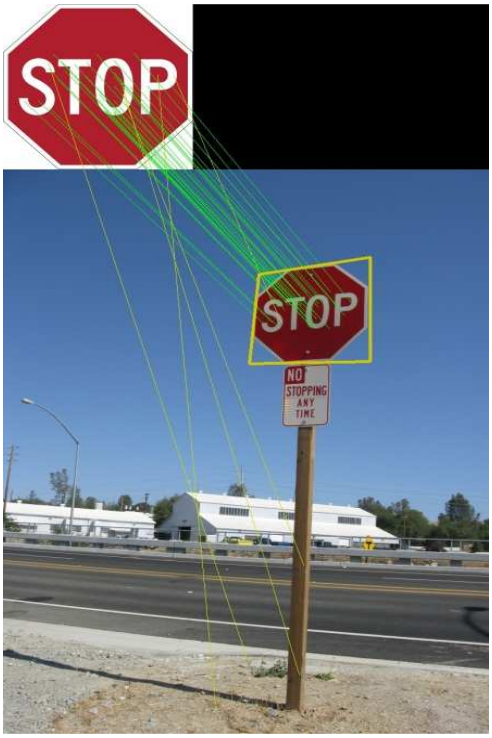

Figure 51 - Stop sign matched from Input1 (top-left) to Input2 (bottom) using features with counts 0-10 (Input1) and count 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [11] [14]

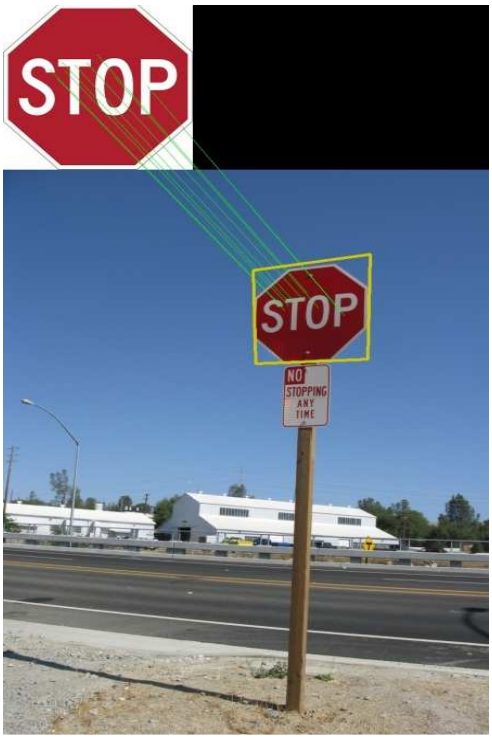

Figure 52 - Stop sign matched from Input1 (top-left) to Input2 (bottom) using features with counts 8-10 (Input1) and count 0-10(Input2). Matches drawn in green (correct) and yellow (wrong) [11] [14]

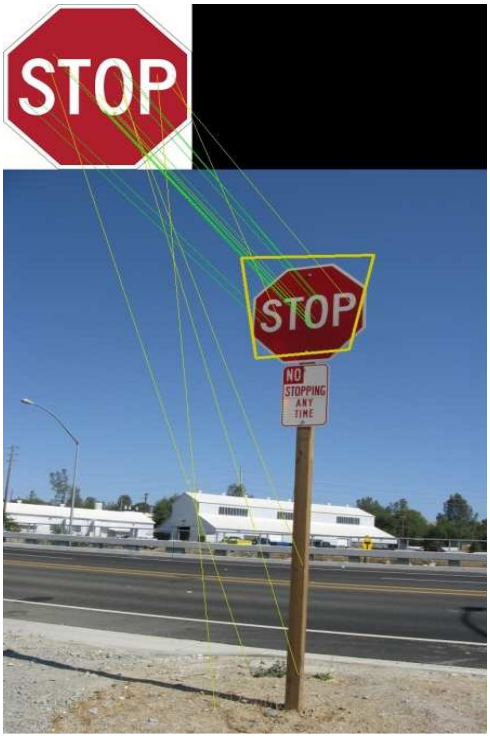

Figure 53 - Stop sign matched from Input1 (top-left) to Input2 (bottom) using features with counts 0-5 (Input1) and count 0-10 (Input2). Matches drawn in green (correct) and yellow (wrong) [11] [14]

In this example, invariant count was used to improve SURF feature matching. Generally SURF features are less when compared to SIFT features on any image. This probably causes our algorithm that uses count for matching to fail sometimes when Input1 is not a subset of Input2. When Input1 is a subset of Input2 our algorithm works well.

From Figure 54 we can say that the features with counts 6-10 are less when compared to features with counts 0-5. In this example we only preprocessed the Input1 and Input2 was not preprocessed. So the range of counts used for Input2 is 0-10 since we used all the features. Whenever features with counts 0-5 are more, we can expect some errors in the matching since more non- robust features are used for matching.

In Figures 54 and 55, we show the count distribution for SURF features. Notice that these are, quite different from our previous SIFT distributions. In particular, these are more uniform and tend to have more feature points in the 0-5 count ranges. This means SURF features are less robust to scale invariance than SIFT features.

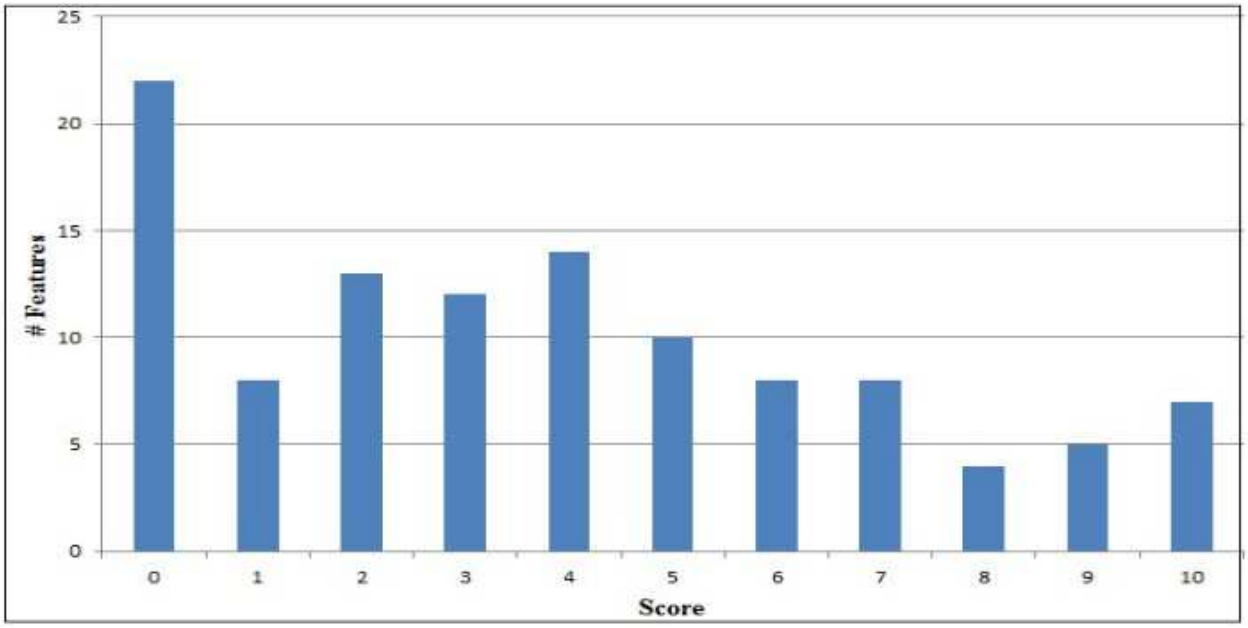

Figure 54 - Number of features per count for Input1 (stop sign)

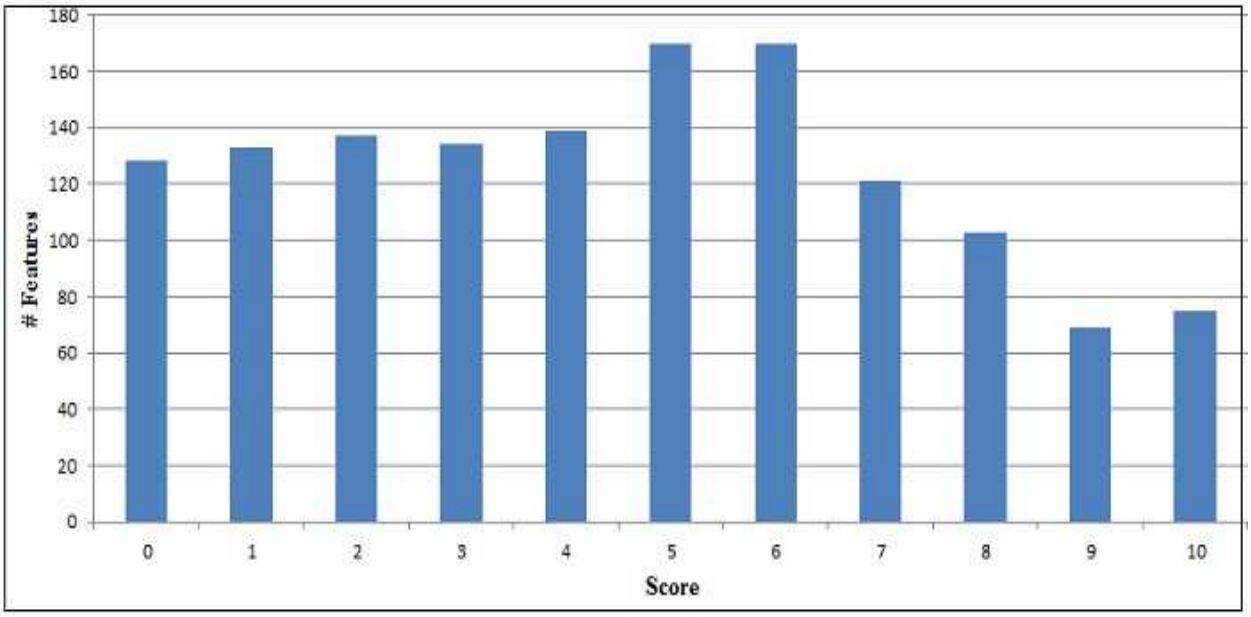

Figure 55 - Number of features per count for Input2

Figures 56, 57 and 58 show the results of 5 matching experiments. In Figure 56 we have total number of matches. In Figure 57 we have percentage of correct matches, and Figure 58 we have the descriptor differences.

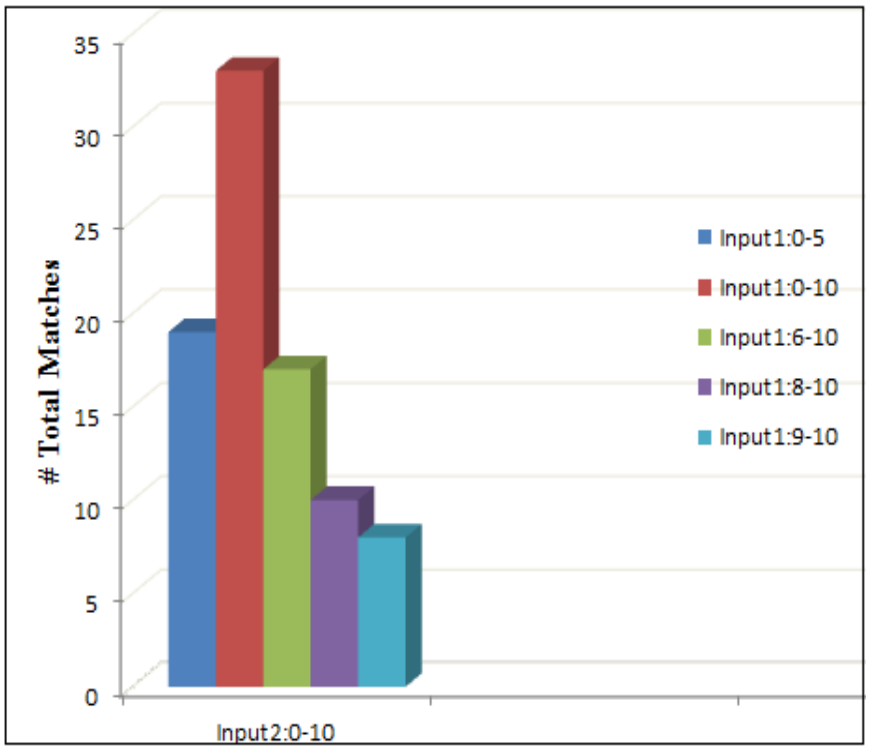

Figure 56 - Total matches between Input1 and Input2. Matching is done using features with different counts from Input1

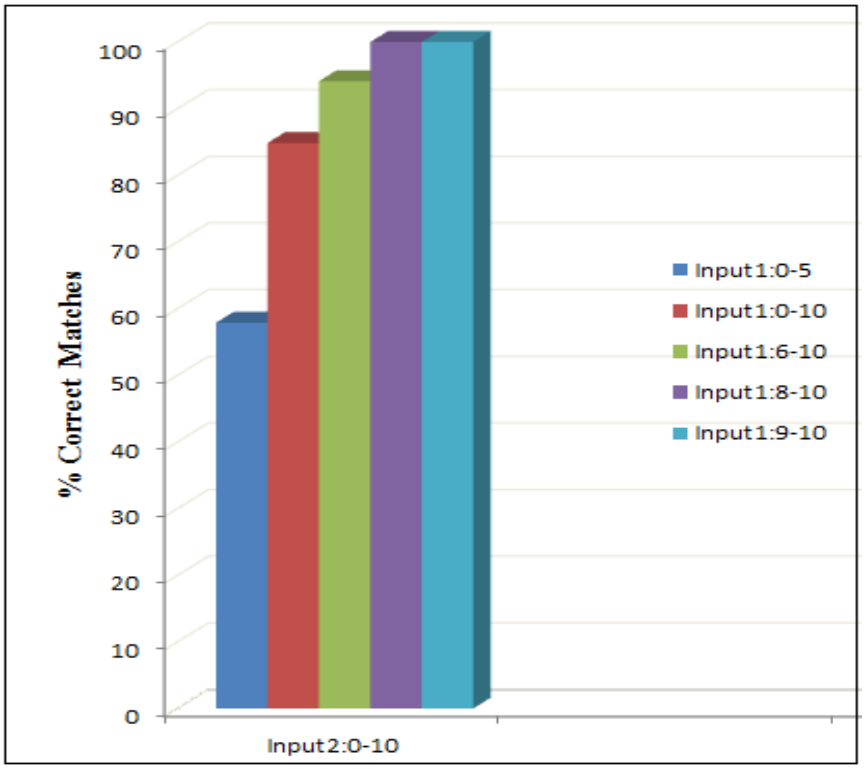

Figure 57 - Percentage of correct matches between Input1 and Input2. Matching is done using features with different counts from Input1

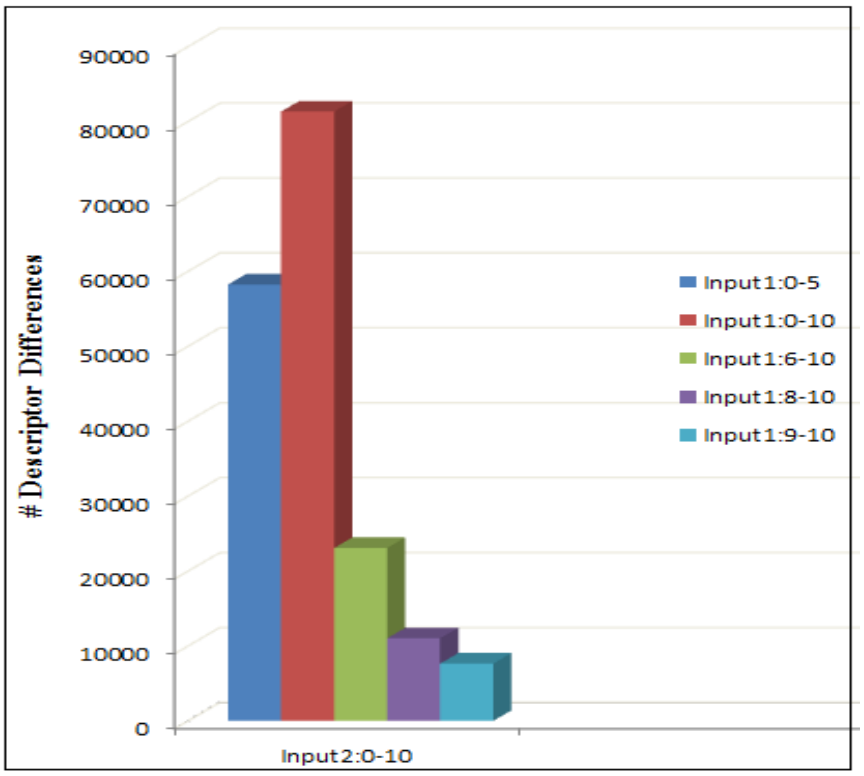

Figure 58 - Number of descriptor differences calculated during the feature matching between Input1 and Input2. Matching is done using features with different counts from Input1

When count range is 0-10 for the Input1 and Input2 the total matches are 33, percentage of correct matches is 85% and the descriptor differences calculated are 81315 (Figure 51). In this case there were 5 errors.

The best count range for matching in this example was 8-10 for Input1 and 0-10 for Input2 (Figure 52). The total matches in this case are 10 and all of them are correct. The percentage of correct matches is 100%. The descriptor differences calculated are 11019. In this case we do not have any errors in the matching and this is after we have used very few features for matching. This certainly shows that preprocessing the stop sign can give us the good results.

The descriptor differences here are 11019, which is 87% less than the previous case where count range is 0-10 for both images.

The worst count range for matching in this example was 0-5 for Input1 and 0-10 for Input2 (Figure 53). In this case the total matches are 19, percentage of correct matches is 58% and the descriptor differences calculated are 58237. Removing the features with counts 0-5 will remove the errors in the matching and reduces matching time.

We can also see how the alignment of the bounding box changes. The count range 0-5 does not give us good alignment. The use of robust features improves or preserves the correct alignment.

6. CONCLUSIONS

Several modern feature detection algorithms use Gaussian scale spaces in order to locate scale-invariant and rotationally invariant feature points in an image, including the Scale-Invariant Feature Transform (SIFT) algorithm. These SIFT features are used to enable a wide range of applications, including object recognition, motion tracking, and image stitching. One problem with SIFT is the fact that the number of features detected in an image can become very large, especially if the input image is big or has a lot of detail. This slows down feature matching and reduces the performance of these applications.

In this thesis, we have demonstrated several different ways to improve feature matching by increasing the quality and reducing the number of SIFT features. Our primary result was an algorithm to identify robust SIFT features by evaluating how invariant individual feature points are to changes in scale. This allows us to exclude poor SIFT feature points from the matching process and obtain better matching results in reduced time. We have also demonstrated that this approach can be applied to SURF features to greatly reduce the number of feature points and improve matching results. We also developed techniques consider scale ratios and changes in object orientation when performing feature matching. This allows us to exclude false-positive feature matches and obtain better image alignment results.

In general image matching can be measured by the following factors: number of features being used in matching, speed and accuracy of our matching algorithm. In our research we are able to improve image matching by improving all the three factors. Number of features used for matching is reduced by identifying robust features. This greatly reduces the post processing time. Robust features certainly improve the matching itself. Even though the preprocessing of the

inputs may take a few seconds, in the end it proves to be worth it. The good thing about our algorithm is it works well with both the features detection algorithms SIFT and SURF.

By using robust SIFT or SURF features, the descriptor difference calculations are at least reduced to half and this speeds up the matching. Even though we use fewer features for the image matching both the alignment and the correct matches are preserved. In fact, in most of the cases the percentage of correct matches increases since the non-robust features are removed.

7. FUTURE WORK

Even though we have been able to reduce the number of SIFT or SURF feature points, the number of descriptor difference calculations is still considerable and needs to be reduced. One approach we could consider is how SIFT and SURF features behave when an image is rotated. These features are intended to be rotation invariant, but as with scale invariance, some points may be more rotationally invariant than others. If we can exclude the points with poor rotational invariance, then we could reduce the number of SIFT and SURF points even more prior to feature matching. Perhaps a counting scheme like the one we devised for scale invariance could be used for this purpose. Another option would be to consider a family of affine transformations (translation, rotation, scale, and skew) and use this to identify the most robust SIFT or SURF feature points.

In our research we have shown how the use of scale ratio window and orientation window improve the matching results. When these windows are calculated using the correct matches from the homography result, the windows are very accurate. Suppose the windows were given even before the actual matching started the descriptor differences calculated are greatly reduced to few hundreds. The idea here is to run the matching algorithm twice, first time to calculate the projected scale ratio and orientation windows using a subset of the feature vectors, and then second time to do the real matching with the full set of feature vectors using these windows. The only problem here is to find a small set of robust features for the first matching.

REFERENCES

[1] D. Lowe, "Object Recognition From Local Scale-Invariant Features," *International Conference on Computer Vision,* Corfu, Greece, September 1999.

[2] D. Lowe, "Distinctive Image Features From Scale-Invariant Keypoints," *International Journal of Computer Vision,* Vol. 2, No. 60, pp. 91-110, 2004.

[3] T. Lindeberg, "Scale-space," *Encyclopedia of Computer Science and Engineering,* vol. 4, John Wiley and Sons, Hoboken, NJ, pp. 2495-2504, 2009.

[4] H. Bay, "SURF: Speeded Up Robust Features," *European Conference on Computer Vision,* Graz, Austria, May 2006.

[5] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 8, No. 6, pp. 679-698, November 1986,

[6] C Harris, "A Combined Corner and Edge Detector," *Proceedings of the Fourth Alvey Vision Conference,* pp. 147-152, September 1988.

[7] A. Vedaldi, "VLFeat.org," 7 December, 2012. http://www.vlfeat.org/overview/sift.html

[8] L. Juan, "A Comparison of SIFT, PCA-SIFT and SURF," *International Journal of Image Processing,* Vol. 3, Issue 4, pp. 143-152, 2009.

[9] OpenCV, "OpenCV.org," 7 December, 2012. http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography

[11] Wikipedia, "wikipedia.org," 7 December, 2012. http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/

[12] Nowpublic, "nowpublic.com," 13 December, 2012. http://www.nowpublic.com/tech-biz/sunday-noon-have-one-dr-pepper-little-gin

[13] Bevreview, "bevreview.com," 13 December, 2012. http://www.bevreview.com/2006/08/28/update-cherry-vanilla-dr-pepper/

[14] Srtctransportation, "srtctransportation.com," 13 December, 2012. http://srtctransportation.blogspot.com/2010_05_01_archive.html

[15] Leafscentral, “leafscentral.ca,” 13 December, 2012. http://lcforum.leafscentral.ca/showthread.php/8517-Look-what-I-bought!/page88

[16] UofA, “uark.edu,” 13 December, 2012. http://www.uark.edu/home/11160.php