SURF: Speeded-Up Robust Features

COMP 558 – Project Report

Presented By: Anqi Xu (260148014) & Gaurav Namit (260307292)

Presented To: Professor Kaleem Siddiqi

McGill University

Due: December 15th, 2008

Table of Contents

Abstract 2
I. Introduction 3
II. Previous Work 4
III. SURF Detector 5
A) Theory 5
B) Implementation 6
C) Discussion 8
IV. SURF Descriptor 9
A) Theory 9
B) Implementation 10
C) Discussion 11
V. SURF Comparator 11
A) Theory 11
B) Implementation 12
C) Discussion 13
VI. Experimentation 13
A) Setup 13
B) Common Failure Modes 14
C) Results as a Function of System Parameters, using CPP-SURF databases 16
D) Results as a Function of System Parameters, using MAT-SURF databases 17
E) Results as a Function of Database Size, using CPP-SURF databases 17
F) Summary 17
VII. Conclusion 18
Bibliography 19
Appendix A: Approximate Nearest Neighbour Algorithm 21
A) Theory & Implementation 21
B) Experimental Setup 22
C) Experimental results 22
D) Conclusion 23
Appendix B: Supplementary Project Information 24
Appendix C: Sample Images in the COIL-100 Library 26
Appendix D: Experiment Result Graphs 27
A) Results as a Function of System Parameters, using CPP-SURF databases 27
B) Results as a Function of System Parameters, using MAT-SURF databases 29
C) Results as a Function of Database Size, using CPP-SURF databases 29

Abstract

For this project, we investigate a recently-conceived image feature detection scheme called SURF, which stands for Speeded-Up Robust Features. This system has been shown experimentally by its authors to outperform the current industry standard detector – Scale-Invariant Feature Transform (SIFT). Motivated by these results, we decided to study SURF by implementing its feature detector and its feature descriptor. We then applied SURF to solve a simplified object recognition application, both to assess this framework's performance, and to compare results between our implementation and the one built by the original authors. In particular, we conducted a series of experiments using the original C++ implementation to determine optimal parameters and ideal detection rates on the entire Columbia Object Image Library (COIL-100) [1], *provided freely by Columbia University. We then carried out a similar set of experiments using our MATLAB SURF implementation on a small hand-chosen subset of the COIL-100 database.*

I. Introduction

Throughout this semester in our *Introduction to Computer Vision* course, we have investigated different computer vision applications, including stereoscopic reconstruction, motion analysis, and object recognition. All the algorithms that we have learned in class analyze images by processing all available pixels directly. Because these so-called "dense" algorithms require high-resolution images (i.e. a large number of pixels) to achieve optimal performance, they generally have relatively slow processing speeds. Additionally, these algorithms are very sensitive to noisy data – since the image noise is present in many pixels, the distortion is significantly amplified when every pixel is processed independently.

In contrast, humans can swiftly and accurately compare and categorize images, even when we are looking at noisy scenes. Our brain can achieve this because we do not work with pixels directly, but instead focus our attention on a *small set of locations* in the image that can uniquely describe the underlying object(s) or scene. These so-called feature regions include corners, edges, and symmetrical patterns. This theory is recognized by neuropsychologists and computer vision researchers worldwide, and is supported by countless experiments performed on mammalian retinal cells and on the visual cortex of primates [2].

Biologically-inspired vision algorithms mimic the human visual system by reasoning about images within the feature space. By summarizing a large number of pixels into a smaller set of features, the source data for these algorithms are hence much smaller in size than in their pixel-based counterparts, which results in faster performance. Additionally, by only choosing features that are relevant to the problem at hand, the input data contains less redundant and useless information, which minimizes erroneous outcomes. Furthermore, because features are extracted from multiple pixels, they can still be recovered even if some pixels are distorted. Thus, pre-processing pixels into features also implicitly filters out noise in the input.

In this report, we present our understanding of a cutting-edge image feature scheme known as Speeded-Up Robust Features (SURF). SURF is comprised of a feature detector based on a Gaussian second derivative mask, and a feature descriptor that relies on local Haar wavelet responses. This framework shares many conceptual similarities with the most widely used feature detector in the computer vision community, called the Scale-Invariant Feature Transform (SIFT) [3]. Interestingly, the authors of SURF have demonstrated experimentally that their new feature scheme outperforms SIFT and other popular feature frameworks, both in terms of speed and accuracy [4][5].

We decided to implement SURF for our Computer Vision course project because we were intrigued by the fact that SURF was able to improve its performance all-around without making any compromises. In particular, we built a version of this feature framework from scratch using MATLAB. To quantify this system's performance, we then employed SURF to solve an object recognition task.

Our approach to object recognition is to compare features in a query image to similar ones found in images within a database. For our data source, we chose the Columbia Object Image Library (COIL-100), which was created at Columbia University [1]. Each photo in this library contains a single object among 100 randomly-chosen articles. Each object is presented in full view and against a dark and plain background, and is positioned in a pre-determined and provided orientation. This library is convenient because each photo focuses on a single object, thus it eliminates the possibility of erroneous recognition results affected by non-static backgrounds. Even though this setup may not reveal all of SURF's technical intricacies, we still expect to be able to explore and quantify many of SURF's algorithmic abilities.

We generated several feature databases using the original authors' C++ SURF implementation (hereby referred to as CPP-SURF), over the entire COIL-100 library. Given our project's time constraints and our trials' slow processing speeds, we then hand-picked 10 structurally-distinct objects from this library, and named it the COIL-10 subset. We processed images in the COIL-10 library through our MATLAB SURF implementation (hereby referred to as MAT-SURF), therefore creating another set of feature databases.

We used each database separately to perform our recognition task. Then, we conducted experiments to determine optimal recognition rates in relation to database-building and feature matching parameters. We also used these results to compare our implementation against the one written by the original authors.

The rest of this report is divided as follows: section II provides a brief review of the literature on various image feature frameworks. Sections III, IV, and V contain detailed analyses of this project's three components: the SURF feature detector, the SURF feature descriptor, and the object recognition feature comparator. Section VI presents our experiments and interprets their results. Finally, section VII concludes this report with a summary and a brief discussion on open-ended topics.

II. Previous Work

SURF's detection scheme is based on the concept of automatic scale selection, proposed by Lindeberg in 1998 [6]. In this work, Lindeberg experimented with using the determinant of the Hessian matrix for a 2-D Gaussian, as well as the Laplacian (i.e. the Hessian's trace), to detect blob-like structures in images. Mainly motivated by Lindeberg's findings, the authors of SURF chose the determinant of Hessian as their target feature. Other well-known feature schemes include the famed Harris corner detector [7] (which relies on eigenvalues of the second moment), the entropy-based salient region detector proposed by Kadir and Brady [8], and the edge-based work of Jurie and Schmid [9]. Furthermore, SURF's detector extends on Lowe's idea of using the Difference of Gaussian as an approximation of the Laplacian of Gaussian filter [3].

A large variety of methods to describe features has been proposed in the literature, including moment invariants [10], steerable filters [11] and phase-based descriptors [12]. However, several studies [13] [14] have shown that Lowe's neighbourhood distribution idea in the SIFT feature framework [15] is by far the most robust descriptor. In recent years, numerous improvements have been proposed for this descriptor [13] [14] [16], but they all involve some type of trade-offs that may cause major hindrances.

The core concept of feature comparison is to find nearest neighbours for a numerical vector. However, given a large feature count and a high vector dimensionality, finding true nearest neighbours can be extremely slow. Beyond the mainstream approximation search algorithms using hash tables and K-dimensional trees (i.e. k-d trees), various improvements have been proposed by the image feature community. These include the Best-Bin-First strategy proposed by Beis and Lowe [17], Omohundro's balltrees [18], Nistér and Stéwenius's vocabulary trees [19], and the work on redundant bit vectors by Datar *et al* [20]. Complementary to these methods, in their original paper, the authors of SURF proposed an indexing method to increase matching speed, using the sign of the Laplacian [4].

SURF's authors have released a closed-source implementation of this feature scheme, written using C++ and without any dependencies [21]. This implementation has since been integrated into the widely-used Open Source Computer Vision library (OpenCV). Prior to its integration, a team in UK used OpenCV to write an open-source C++ version of SURF, based on the original authors' publications [22].

III. SURF Detector

A) Theory

The SURF detector focuses its attention on blob-like structures in the image. These structures can be found at corners of objects, but also at locations where the reflection of light on specular surfaces is maximal (i.e. light speckles). In 1998, Lindeberg noticed that the Monge-Ampère operator and Gaussian derivative filters could be used to locate features. Specifically, he detected blobs by convolving the source image with the determinant of the Hessian (DoH) matrix, which contains different 2-D Gaussian second order derivatives. This metric is then divided by the Gaussian's variance, σ^2, to normalize its response:

$$DoH(x,y,\sigma) = \frac{G_{xx}(x,y,\sigma)\cdot G_{yy}(x,y,\sigma) - G_{xy}(x,y,\sigma)^2}{\sigma^2} \qquad where\ G_{ij}(x,y,\sigma) = \frac{\partial N(0,\sigma)^2}{\partial i \cdot \partial j} * image(x,y)$$

The local maxima of this filter response occurs in regions where both G_{xx} & G_{yy} are strongly positive, and where G_{xy} is strongly negative. Therefore, these extrema occur in regions in the image with large intensity gradient variations in multiple directions, as well as at saddle points. Visually, this means that blob-like structures refer to corners and speckles.

The other reason why many feature detection schemes rely on Gaussian filters is to get rid of noisy data by blurring the image. As a side-effect, Gaussian blurring highlights image details at or near a single unique scale. As Marr wrote in the *Theory of Edge Detection*, "no single filter can be optimal simultaneously at all scales, so it follows that one should seek a way of dealing separately with the changes occurring at different scales." [23] He goes on to propose to extract features using the combined information of multiple responses, generated using the same family of filters but at different scales. This resulting stack of convolutions is typically referred to as the scale space.

SURF's authors and many others have found that the number of results in scale-space decreases exponentially with increasing scale size. One potential explanation is that large amounts of Gaussian blurring can average out nearly all useful information in images. Therefore, searching through the scale space linearly can be very wasteful computationally. As an alternative, SURF introduces the notion of scale octaves: each octave linearly samples a small region of the scale space, but this region size is proportional to the scale magnitudes used. Each adjacent octave doubles the region size and sampling increment used in the previous one, to reduce the amount of search necessary at larger scales.

The only negative side-effect of using octaves is that results found at larger scales can potentially have more error, due to the larger increments used in their corresponding octaves. To compensate, the SURF detector interpolates the coordinates of any local maxima found into the sub-pixel and sub-scale range.

Finally, SURF's authors propose to make the distinction between bright blobs found on a dark background, versus dark blobs found on a bright background. This property can be represented by the sign of the Laplacian, as shown below:

$$sgn\{G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)\} = \begin{cases} +1 & \Rightarrow \quad bright\ blob\ over\ dark\ background \\ -1 & \Rightarrow \quad dark\ blob\ over\ bright\ background \end{cases}$$

The SURF detector algorithm can thus be summarized by the following steps:

1. Form the scale-space response by convolving the source image using DoH filters with different σ
2. Search for local maxima across neighbouring pixels and adjacent scales within different octaves
3. Interpolate the location of each local maxima found
4. For each point of interest, return x, y, σ, the DoH magnitude, and the Laplacian's sign

B) Implementation

To find blobs, we need to convolve the source image with various Gaussian-related filters. Because continuous Gaussian filters have non-integer weights, these convolutions require floating-point operations, which can severely hamper processing speed. Thus, SURF's authors propose to approximate these filters by using box-shaped, integer-weighted masks, as shown in Fig 1.

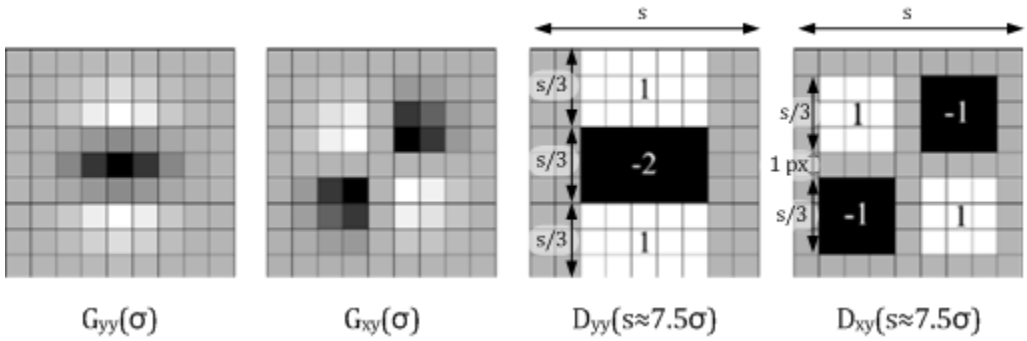

Fig 1: Gaussian second derivatives $G_{yy}(\sigma)$ & $G_{xy}(\sigma)$ and box filters $D_{yy}(s)$ & $D_{xy}(s)$ (image modified w/o perm., from [5])

$$DoH(x,y,\sigma) \approx \frac{D_{xx}(x,y,\sigma)\cdot D_{yy}(x,y,\sigma) - \left[\beta \cdot D_{xy}(x,y,\sigma)\right]^2}{\sigma^2}$$

$$where\ \beta = \frac{\left\|G_{xy}(x,y,\sigma)\right\|_F \cdot \left\|D_{yy}(x,y,\sigma)\right\|_F}{\left\|G_{yy}(x,y,\sigma)\right\|_F \cdot \left\|D_{xy}(x,y,\sigma)\right\|_F} \approx 0.9$$

A compensation term β is inserted in the computation of DoH to compensate for the error introduced by approximating the true Gaussian derivative masks. This multiplicative factor defines the ratio of Frobenius norms $\left(\|A(x,y,\sigma)\|_F \equiv \sqrt{\sum_{x,y}|A(x,y,\sigma)|^2}\right)$ between the box filter approximations and their true counterparts. Even though β is dependent on the scale size σ, it turns out that in practice β can be approximated using a static constant of 0.9.

By using filters with rectangular regions, SURF can also take advantage of a swift convolution technique using integral images. The value at each pixel in an integral image is the cumulative sum of intensities within the rectangular region, bounded by opposite corners that are located at the image's origin and at the current coordinate. As Fig 2 illustrates, any rectangular-shaped convolution response can be obtained by adding and subtracting 4 values in the integral image. This means that the scale-space response can be computed in *constant time*: rather than blurring the image multiple times, we only need to look up a constant number of entries in a *single* integral image, while choosing positions based on their scale.

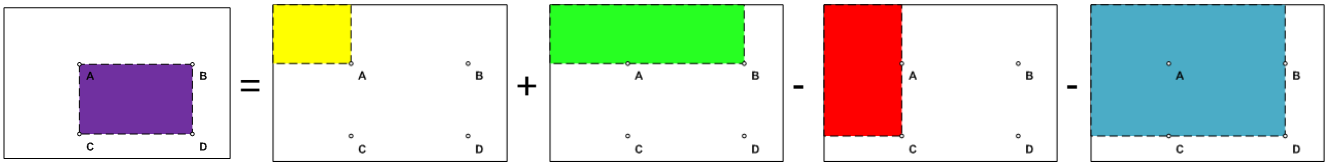

Fig 2: Demonstration of obtaining the sum of intensities within any rectangular region by using 4 integral image values

Each octave is composed of 4 box filters, which are defined by the number of pixels on their side (denoted by s). As stated in Fig 1, s is simply a scaled version of the Gaussian function's standard deviation, σ. The first octave uses filters with 9x9, 15x15, 21x21 and 27x27 pixels (i.e. s = {9, 15, 21, 27}, respectively). The second octave uses filters with s = {15, 27, 39, 51}, whereas the third octave employs values of s = {27, 51, 75, 99}. If the image is sufficiently large, a fourth octave is added, for which s = {51, 99, 147, 195}. These octaves partially overlap one another to improve the quality of the interpolated results.

To obtain local maxima of the DoH responses, SURF's authors propose to employ a Non-Maximum Suppression (NMS) search method with a 3x3x3 scanning window. Neubeck and Van Gool suggested several algorithms to compute NMS efficiently [24]. We chose a very simple block-based approach, which takes advantage of the fact that there can be *at most* 1 maximum value within any 2x2x2 volume, when the search window is 3x3x3. To prove this, consider a 1-Dimensional case where we look for local maxima positions within a 3-pixel neighbourhood. When we look at any 2 consecutive pixels, either one of them is larger than the other (thus resulting in a local maximum), or they have identical values (in which case we assume that neither entries are maxima).

Using this fact, block-based NMS looks for the maximum value within each non-overlapping 2x2x2 region. When a candidate is found, we already know that 7 of its 26 neighbours have smaller values, because these 7 pixels lie within the original searching volume. Therefore, to confirm that the candidate is indeed the 3x3x3 local maximum, we only need to make 19 additional comparisons. Intuitively, this algorithm reduces the number of comparisons by up to a factor of 8, even though it is still trivial to implement!

Features with large DoH values are most suitable to represent their underlying objects, because they are less likely to appear in a random image. In contrary, maxima with small DoH values can potentially arise from imaging noise and other artefacts, and thus are not very useful at discriminating between different objects. Therefore, the SURF algorithm employs a detector cut-off threshold, T_D, to reduce the number of these weak features. However, we must be careful not to set T_D to be too large, or else the algorithm might not return a sufficient number of features.

As we mentioned previously, once we have located a set of local maxima within each octave, we need to interpolate their positions and scales. We use the method suggested by Brown and Lowe [25], and fit a 3D quadratic to the local DoH neighbourhood using a 2nd order Taylor series approximation:

$$DoH(\boldsymbol{x}) \approx DoH(\boldsymbol{x_0}) + \frac{\partial DoH(\boldsymbol{x})^T}{\partial \boldsymbol{x}} \cdot \boldsymbol{x} + \frac{1}{2} \cdot \boldsymbol{x}^T \cdot \frac{\partial^2 DoH(\boldsymbol{x})}{\partial \boldsymbol{x}^2} \cdot \boldsymbol{x} \qquad where\ \boldsymbol{x} = \begin{pmatrix} x \\ y \\ \sigma \end{pmatrix}$$

We obtain the interpolated position and scale by differentiating the previous equation and equating it to 0:

$$\boldsymbol{x}_{max} = \left[\frac{\partial^2 DoH(\boldsymbol{x})}{\partial \boldsymbol{x}^2}\right]^{-1} \cdot \left[\frac{\partial DoH(\boldsymbol{x})}{\partial \boldsymbol{x}}\right]$$

In our implementation, we used a first order central difference technique to calculate discrete derivatives:

$$\left.\frac{\partial f(\dots, x, \dots)}{\partial x}\right|_{x_0} \approx \frac{f(\dots, x_0 + 1, \dots) - f(\dots, x_0 - 1, \dots)}{2}$$

To increase the system's robustness, the original authors recommend repeating the algorithm on a version of the input image that is up-scaled by a factor to 2. Due to the increased input size, each octave in this setting contains 5 scales instead of 4, and they are: s = {15, 21, 27, 33, 39} for the first octave, s = {21, 33, 45, 57, 69} for the second octave, s = {33, 57, 81, 105, 129} for the third octave, and s = {57, 105, 153, 201, 249} for the optional fourth octave. However, we noticed that many features obtained previously were duplicated by processing the up-scaled image. Since these double entries have slightly different coordinates, we could not find a general method to detect and remove them that was independent of the setup and data source. Additionally, we also observed that the up-scaled image took 4 times longer to compute. Therefore, we decided not to use this feature in our experiments.

C) Discussion

When we began testing our implementation, we soon realized that using a static 3x3x3 NMS window to find the local maxima is fundamentally flawed. In particular, because larger-sized filters tended to smooth out details among neighbouring data, we observed that the local maxima sometimes *drifted* in an arbitrary direct as we tracked its location across different scales.

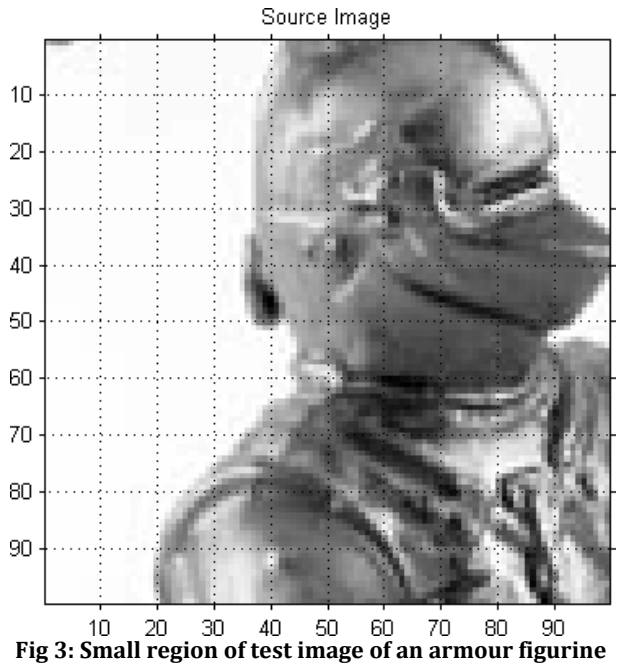

Fig 3: Small region of test image of an armour figurine

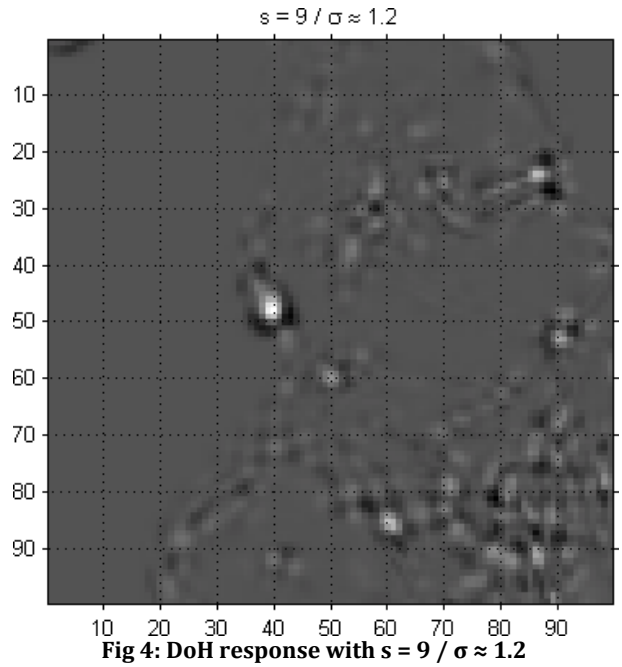

Fig 4: DoH response with s = 9 / σ ≈ 1.2

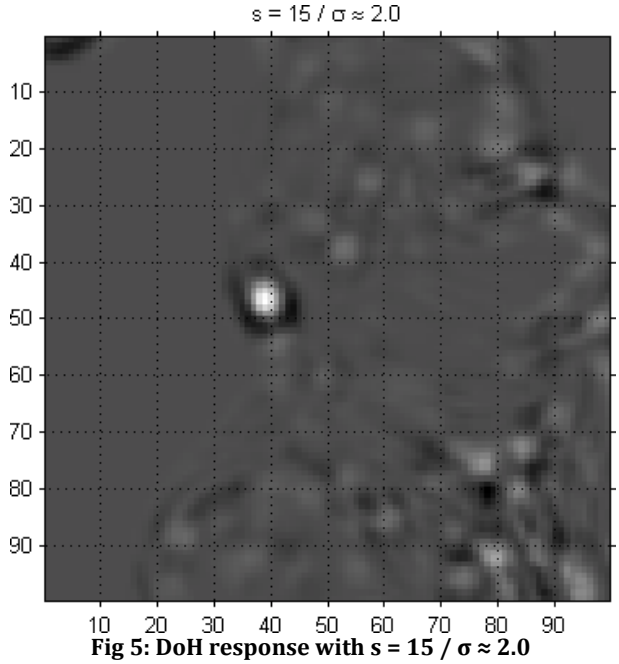

Fig 5: DoH response with s = 15 / σ ≈ 2.0

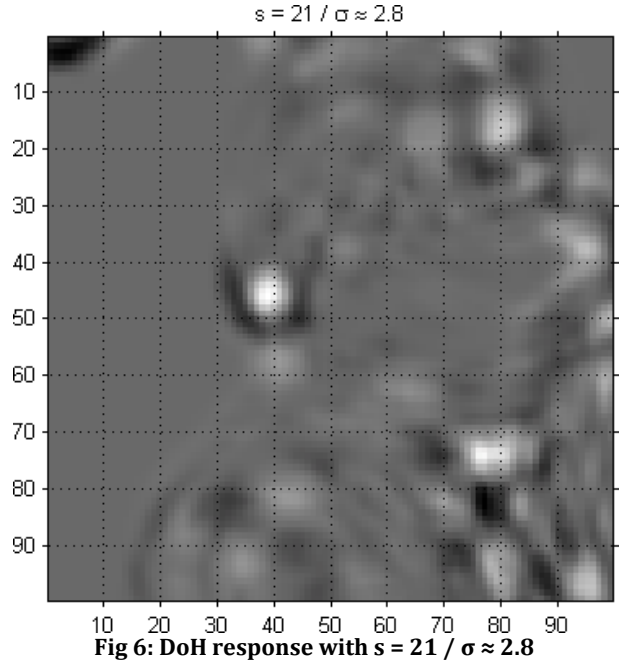

Fig 6: DoH response with s = 21 / σ ≈ 2.8

To illustrate our point, we present a test image of an armour figurine in Fig 3, where one major blob is located at the wedge-shaped structure on the back of the helmet. As we track its position in the DoH responses generated using the first 3 scales (s = 9, 15, 27) of the first octave, as shown in Fig 4 through Fig 6, we note that the white regions corresponding to the local maxima moves from of (49, 39) at s = 9 to approximately (45, 36) at s = 21. Therefore, our implementation cannot capture these location changes by using a 3x3x3 scanning window. Because this phenomenon is aggravated at larger scales, in general our detected feature points often have slight offsets. Strangely, CPP-SURF did not appear to have this issue.

We also observed another problematic phenomenon in the interpolation step, which can be partially explained by the inability of the 3x3x3 NMS window to track features across multiple scales: in some situations, the quadratic fit attains its maximum at a location outside of the specified 3x3x3 region. We believe that these deviations are caused by numerical errors originating from using discrete derivatives. We cannot use these interpolated values to identify features, both because they might not be accurate and because the suggested locations might lie outside of the image's boundaries. To resolve this issue, we perform quadratic fitting to the 3 parameters (i.e. x, y, and s) independently. Although this work-around yields poorer results, at least we are not forced to discard these feature candidates altogether.

Finally, when we tested our MATLAB implementation on random images, we saw that imaging conditions and environmental settings sometimes caused identical features to have different Laplacian signs. For example, we observed that when a light grey object was photographed while lit by a white light source, and then captured again while lit by a dark blue light source, speckles at identical locations on the object's surface in these two images had opposite Laplacian signs. Therefore, we decided to ignore the sign of the Laplacian when comparing features in our object recognition task.

IV. SURF Descriptor

A) Theory

To describe each feature, SURF summarizes the pixel information within a local neighbourhood. The first step is determining an orientation for each feature, by convolving pixels in its neighbourhood with the horizontal and the vertical Haar wavelet filters. Shown in Fig 7, these filters can be thought of as block-based methods to compute directional derivatives of the image's intensity. By using intensity changes to characterize orientation, this descriptor is able to describe features in the same manner regardless of the specific orientation of objects or of the camera. This rotational invariance property allows SURF features to accurately identify objects within images taken from different perspectives.

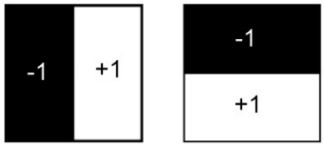

Fig 7: Horizontal and vertical Haar wavelet filters

In fact, the intensity gradient information can also reliably characterize these pixel regions. By looking at the normalized gradient responses, features in images taken in a dark room versus a light room, and those taken using different exposure settings will all have identical descriptor values. Therefore, by using Haar wavelet responses to generate a unit vector representing each feature and its neighbourhood, the SURF feature framework inherits two desirable properties: lighting invariance and contrast invariance.

Because the neighbourhood can be broken down into smaller windows in arbitrary manners and because the individual Haar wavelet responses can be summarized differently, multiple variants of SURF have been proposed using different combinations of settings. Additionally, in applications such as mobile robotic vision, images are captured using statically-positioned cameras, so these photos are all oriented in similar directions. To save computation speed, a variant of the algorithm called U-SURF (where U stands for up-right) foregoes the orientation step, and computes the horizontal and vertical Haar wavelet responses using the cardinal axes of the image directly. The original authors have experimentally demonstrated that U-SURF features are orientation-invariant up to 15°.

B) Implementation

The Haar wavelet is composed of two rectangular regions, which means that integral images can be used to compute these convolutions efficiently. However, similar to defects discussed previously in part C of section III, computing discrete derivatives can be severely hampered by image noise. Thus, a Gaussian distance mask is used to reduce the contribution of far-away pixel entries from each feature's center point.

To compute the orientation, two directional Haar wavelet filters of size 4σ are convolved with a 12σ x 12σ square region positioned about each feature location. After weighting the responses at each sampling pixel by a Gaussian distance factor (with a standard deviation of 2σ), they are interpreted as points in a 2-Dimensional plane. The algorithm then looks for a continguous region spanning 60° within this plane, for which the sum of all the horizontal and vertical responses are maximal. Finally, the center of the 60° window containing the strongest response is recognized as the dominant orientation.

We implemented the scanning window step as follows: first we put the absolute values of the horizontal response (d_x) and the vertical response (d_y) into one of 360 bins, whose index is computed as $\theta(d_x, d_y) = floor[\tan^{-1}(d_y/d_x)]$. This vector of bins is then convolved with a unit rectangular filter of length 60, using an implementation similar to integral images. Finally, we search for the index with largest response, and interpret this index as the angle of the dominant orientation direction.

Once the orientation direction has been determined, we use it to form a 20σ x 20σ square region around the feature, and extract all pixels within that region. We then compute the integral image for this cropped region, and use the method discussed in part B of section III to convolve the data with the horizontal and vertical Haar wavelet filters, each with a size of 2σ. To reduce the effects of geometric deformations and localization errors, all d_x and d_y responses are multiplied by a Gaussian weight (with a standard deviation of 3.3σ) using the distance between each pixel in this region and the center point.

Next, the resulting 400 responses are partitioned into 4x4 sub-windows, thus resulting in 16 sets of 25 d_x and d_y values. This structure is illustrated in Fig 8, where only 4 vectors are shown for each of the 16 sub-windows (instead of 25) for sake of cleanliness. For each sub-window we compute the sum of values ($\sum d_i$) and of magnitudes ($\sum|d_i|$) for both d_x and d_y. Thus, each normalized feature vector has 64 entries.

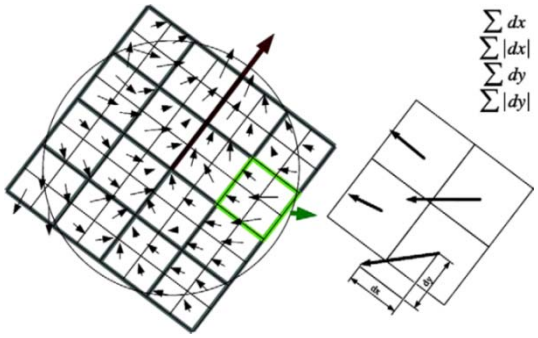

Fig 8: Structure breakdown of each feature's neighbourhood (image borrowed w/o perm., from [5])

A variant called SURF-128 separates the d_x and d_y vectors into two subsets based on the sign of entries, thus yielding 8 metrics per sub-window. Unfortunately, this variant has been demonstrated experimentally not to boast any acceptable benefits that are worth doubling the dimensionality. Another variant called SURF-36 partitions the 20σ x 20σ neighbourhood into 3x3 sub-windows. In general, these features tend to have less discriminative power, but they can be compared very swiftly given their low dimensionality. However, because the primary goal of this project to explore the main properties of SURF, we decided to limit our experiments using only the original SURF-64 descriptor and the U-SURF variant.

C) Discussion

We have identified several aspects of our MATLAB framework that might differ in implementation from the original CPP-SURF system. First of all, recall that the detector interpolates the location and scale for each feature, thus in general these 3 parameters all have non-integer values. When we used these non-integer values to sample feature neighbourhoods, we decided not to perform any type of interpolation, and instead opted for a faster and simpler nearest-neighbour sampling approach. As result, each and every sampled pixel has some associated error, and their combined effect might be noticeably detrimental.

As a footnote in their original paper, SURF's authors suggested that the Haar wavelet responses of an oriented (i.e. non-up-right) neighbourhood can be obtained using the un-rotated image through interpolation. However, we could not figure out how interpolation can be achieved, and thus explicitly re-sampled pixels within the rotated region in our descriptor. We suspect that the effects of this difference can be noticeable in our experimental results.

In general, because SURF's authors left out many technical details in their published works, we had to make some assumptions in our own implementation. In fact, our entire system is based on a major assumption: the use of Haar wavelet convolutions to describe SURF features. In fact, the original authors never explicitly suggested convolving the image with Haar wavelet filters, but rather always referred to the process using the cryptic "Haar wavelet response" term. Unfortunately, in many other fields, the Haar wavelet is often used to create the Haar transform, which maps a temporal signal into a temporal-frequency space. However, after extensive investigation and experimentation, we decided not to implement the Haar transform, and instead convolved the image with constant-sized Haar wavelet filters. Nevertheless, we cannot be absolutely certain that this assumption is correct, as our inquiries sent to the original authors of SURF have not been answered at the time of writing this report.

V. SURF Comparator

A) Theory

We begin this section by formally defining the setup for our object recognition application: we are given an image library of labelled objects, where each image contains a single object in full view without obstruction, over a plain dark background. Each object is represented by multiple images, which are taken from slightly different viewpoints. They are generated using a fixed camera, taking pictures of the object as it sits on top of a controlled turntable. The object's identity and orientation are provided to the user.

The approach to object recognition is to compute a database of features for each image in the library. When given a query image at runtime, we generate the set of query features and attempt to match it to other sets within the database. Once we find the database object with the best feature matching based on some comparison metric, we conclude that this object is present in the query image.

This type of approach is very similar to appearance-based identification algorithms, using eigenspace and Principal Component Analysis (PCA) [26]. With an identical database setup, pixel-based PCA methods compute a high-dimensional vector per image based on the eigenspace of the pixel covariance matrix. Therefore, each object can be represented by a manifold in the high dimensional space of the principal component weights. To match objects using PCA, the query image is projected onto the principal components of each database object's eigenspace, and the associated weight vector is compared to existing manifolds. The object identity of the closest manifold is assigned as the recognition match.

By using features, each image is now represented by a set of feature vectors rather of a single weight vector. Because individual feature descriptor values are not related numerically, each object is no longer represented by a manifold, but instead corresponds to a cloud in feature space containing the features belonging to all of its images. Similarly, the query data can also be seen as a cloud of features. We use the Euclidean distance metric of the nearest neighbouring feature to assess the distance between point clouds.

Our comparator framework takes advantage of several of SURF's key properties. For example, we work solely with the descriptor data, while completely disregarding information returned by the detector. In particular, because the scale values are ignored, features on the same object in two differently-scaled images can be matched to each other. By ignoring the location in the image, we can detect objects even if they are translated in some images. Additionally, because contrast information is normalized by using unit-length vectors, our system can accurately compare images taken under different lighting conditions. Finally, features vectors generated using SURF-64 have very similar values when extracted from rotated versions of the same image. Therefore, this application makes use of SURF's powerful attributes, including scale invariance, translation invariance, lighting invariance, contrast invariance, and rotation invariance.

B) Implementation

In section III, we mentioned that the authors of SURF proposed to distinguish between bright blobs and dark blobs, and do not match between these two feature types. We commented that this restriction can reduce recognition performance in certain circumstances such as images generated using different lighting conditions. Therefore, we decided not to use the sign of the Laplacian to restrict feature matching. Effectively, we have sacrificed processing speed for higher recognition rates. This choice is a direct consequence of our primary goal, which is to explore SURF's ability to represent objects efficiently.

When looking for the closest neighbour for a query feature, a recurring type of situation often occurs where many neighbours within a single database image have comparable distances to the query. In these situations, choosing the specific database feature using Euclidean distance might not always result in a correct matching. These types of pairs increase the likelihood of erroneous recognition because of the ambiguity of their true neighbours. As a conceptual example, if regions with constant color are detected as features, then these vectors will be present in almost all images, and therefore we cannot and should not use these vectors to distinguish objects from one another. It turns out that our so-called weak SURF features (i.e. those with small DoH magnitudes) are most prone to be susceptible to this problem.

To compensate, the matching algorithm will discard a query feature if the ratio of Euclidean distances between its closest neighbour and its second closest neighbour is above a limiting value. This cut-off value is referred to as the comparator threshold, T_C, and it ranges from 0 (exclusive) to 1 (inclusive). When T_C is small, only features that are very unique to their underlying objects are pair up due to their strong discriminating power. On the other hand, if T_C is near 1, the algorithm then allows less discriminating pairs to also affect the recognition process, which might end up hampering the overall performance.

Using the algorithm described above, we can generate a set of feature pairs between the query image and each individual database image. For each image pair, we compute a recognition score, which is defined as the number of feature pairs found (N_i), over the sum of their squared distances (d_{ij}^2):

$$S_R = \frac{N_i}{\sum_{j=1}^{j=N_i} d_{ij}^2}$$

S_R is maximized when we find a large number of pairs all having very similar descriptor values between the query and a database image. Naturally, these two images are most likely to contain the same object.

C) Discussion

In a practical implementation, finding the exact nearest neighbour for a large number of high-dimensional vectors can be incredibly slow. Therefore, deployment-grade feature frameworks can only afford to look for approximate nearest neighbours when used in real-time applications. However, because speed is not one of our project design goals, we conducted all of our experiments using exact feature neighbour matches. Nevertheless, we have also thoroughly investigated the design, implementation and performance of approximate nearest neighbours using K-dimensional trees and Best-Bin-First search [17]. We present this as a separate topic, in Appendix A.

We have made a slight adjustment in our comparator algorithm, which was presented in the previous sub-section. Specifically, we defined the recognition score S_R between the query image and a database *image*, rather than a database *object*. We perform image versus image (IvI) comparisons because we observed that the same features are typically found in images with relatively similar viewing directions, and thus we can configure our system to detect object identities *as well as* image orientations!

On the other hand, in our Theory sub-section, we stated that the system should perform image versus object (IvO) comparisons. Since many features will be present in multiple image views of the same object, we can increase our comparator's processing speed by grouping all features belonging to the same object into a single cloud. Even though we did not implement this setting due to time constraints, we did however use a very similar concept in one of our code variants. Specifically, instead of grouping the features directly, we still found feature pairs between the query image and individual database images. However, we then group these feature pairs based on the respective database objects, and computed the recognition score using these larger sets of pairs.

VI. Experimentation

A) Setup

We employed Columbia University's COIL-100 library as our data source. This library contains a total number of 7200 image representing 100 small-sized objects. The objects are chosen in a seemingly random manner, and include articles from toy cars to dental floss casings. Each object is photographed on a horizontal turntable 72 times, where consecutive photos are taken after adjusting the turntable by 5° increments. We form our query set by taking photos taken at 0° and at 180°, whereas we insert the other 70 images per object into the training database.

We generated feature databases by processing all 7200 images through CPP-SURF. We chose two detector thresholds (T_D = 10, and 100) to assess the effects of this parameter on the recognition performance. On the other hand, because we have previously identified potential theoretical flaws and operational differences between our MAT-SURF and the original CPP-SURF systems, we decided to reduce the image library used by MAT-SURF, by hand-picking 10 visually distinct objects. We named this resulting subset of 720 images as the COIL-10 library, which we illustrate in Appendix C. To generate feature databases, we used four detector threshold values (T_D = 0.01, 0.1, 1, and 5). Please note that the detector threshold is not scaled identically between CPP-SURF and MAT-SURF, due to a negligible difference in the implementation of the DoH filter responses.

We decided to only use the oriented features for the original authors' implementation (i.e. SURF-64). On the other hand, we then conducted similar trials using SURF-64 and U-SURF for our MATLAB-based code.

During our trials, we processed each database through our object recognition algorithm multiple times, with 10 different comparator thresholds ranging between 0.1 and 1 in linear increments of 0.1. We also tested the performance of the system twice, when we compared query images to database objects, and then when we compared queries to individual images in the database.

Most of our trials were generated using different permutations of the aforementioned parameter settings. However, we also examined briefly the possibility and effects of reducing the training database size, by limiting images based on increasingly-large angle increments.

We used three different metrics to assess the performance of our recognition system. First and foremost, because we know the object identity for query images, we can compute the recognition success rate resulting from each individual setup. However, when we first began looking for reasons why we were obtaining low success rates using certain parameter settings, we identified two recurring trends: either the number of feature pairs was very small, or the correct object had a recognition score that was slightly worse than the incorrect candidate with the maximum recognition score. Thus, we used the average number of feature pairs for each unique query-training image pair as our second assessment metric. Furthermore, our framework computed the position of the correct object in descending order of S_R within each setting. We found insights by looking at the minimum, average and maximum values of these positions, but we ultimately chose to use the average position of the correct object as our third and final comparison metric. As a reminder, we do not assess performance speed at all in this project.

We encourage interested readers to repeat our experiments and perform further analysis on our raw results. Appendix B contains various links to our source code and other required material, and also elaborates on different usages of our framework.

B) Common Failure Modes

On the next page, Fig 9 and Fig 10 illustrate a recurrent failure mode of the system, where commonly-seen features (such as the round corners in our example images) are erroneous matched between different objects. As the figure titles indicate, the recognition score between the false image match and the correct one have very similar recognition scores. However, we can immediately observe that the number of feature pairs used in the incorrect match is considerably less than the feature pair count between images of the same object. Even though we chose to illustrate a drastic example, we have observed in our results that the same type of problems occur with nearly all of the incorrectly-recognized images.

To improve performance of our object recognition system, we need to address two issues: first, we need to emphasize the importance of the number of feature pairs when we make our recognition. Since this number is already present in the recognition score, one potential improvement would be to raise its value to a larger power when computing S_R. This change will severely penalize image matches with few feature pairs. The more elaborated version of the previous suggestion is to look at the number of feature pairs for a specific number of leading matches. If we find that the distribution of values is concentrated at the two extremes, then we can boost the recognition score for all occurrences with large values.

The other reason why we have such low feature pair counts, regardless of system parameters, is due to the fact that images in the COIL-100 library all have a very limited resolution of 128 x 128 pixels. Due to this restriction, we are not able to recover enough small details of each object to sufficiently differentiate between different articles. This issue can be solved by using a database with higher quality images.

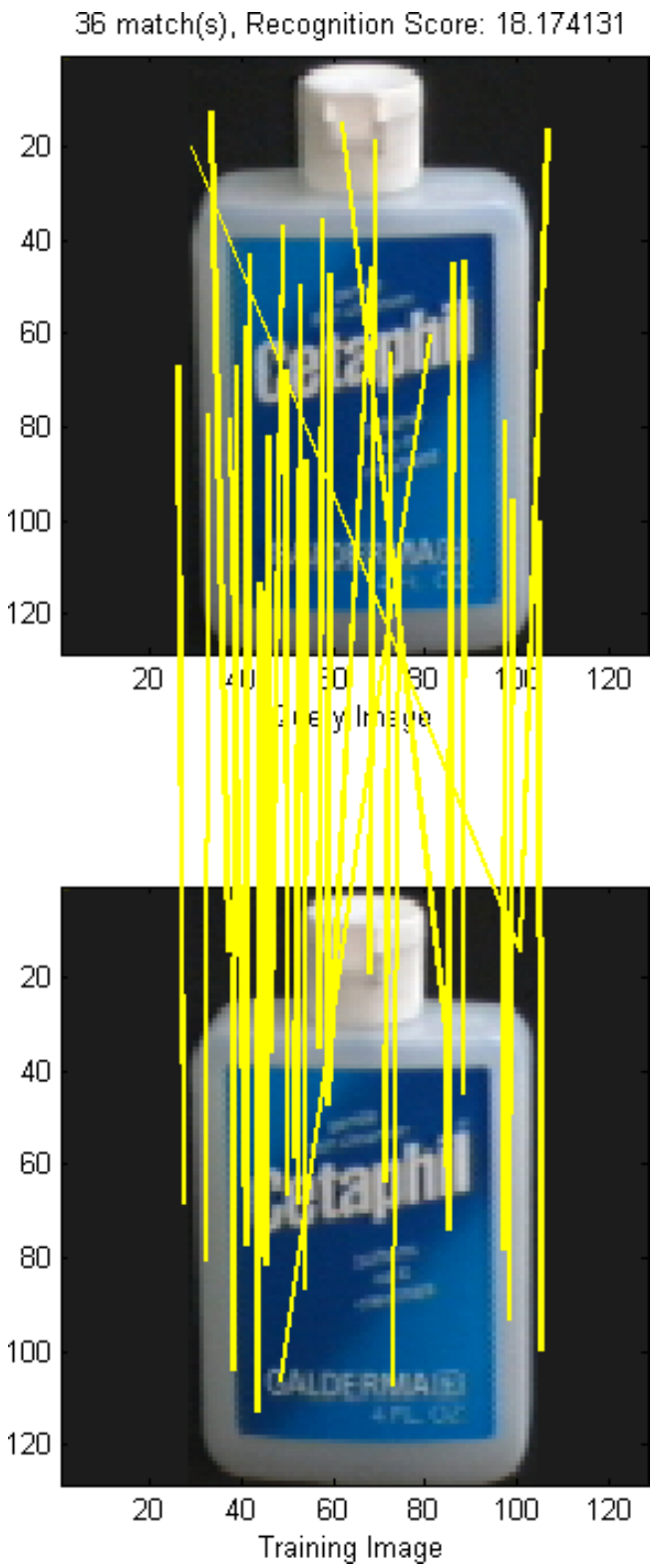

Fig 9: Feature pairs found between object 13 at 0° (upper image) and object 13 at 5° (lower image), generated using CPP-SURF, with $T_D = 10$ and $T_C = 0.8$

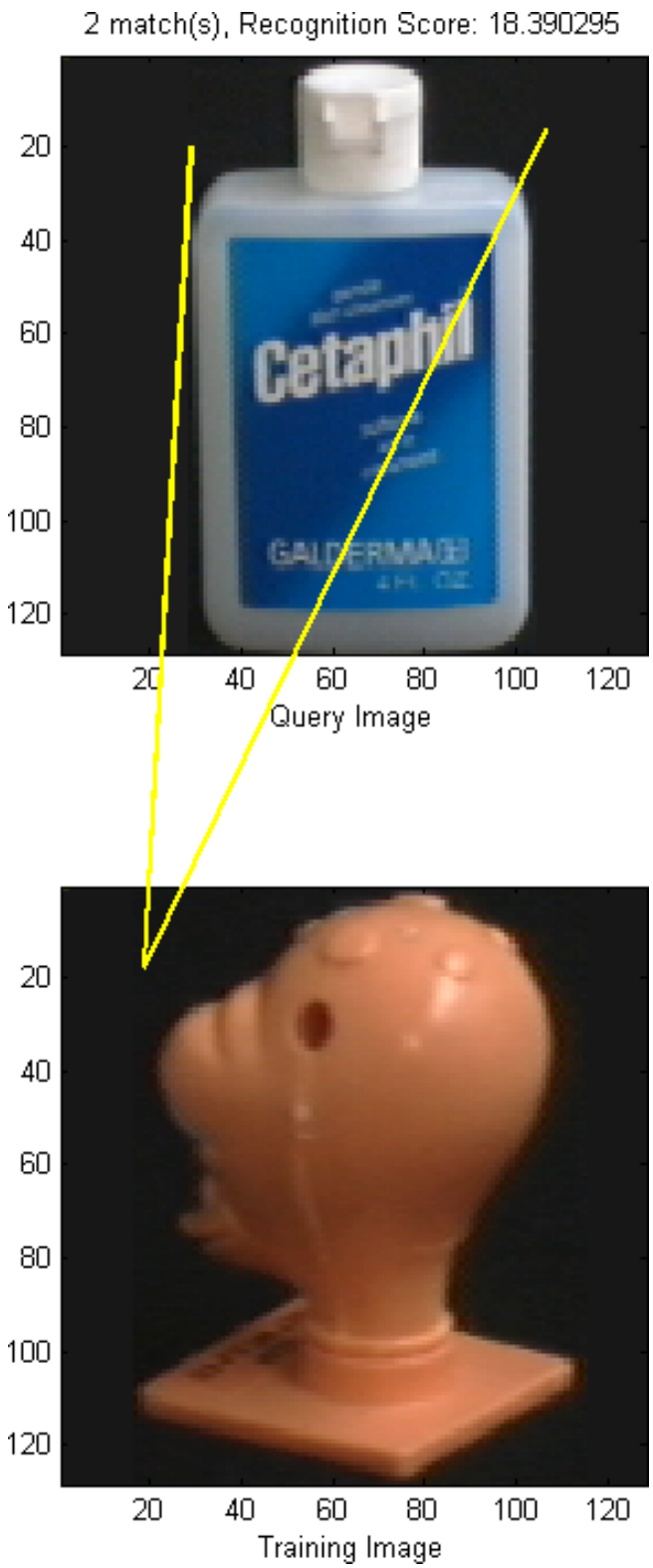

Fig 10: Feature pairs found between object 13 at 0° (upper image) and object 20 at 125° (lower image) , generated using CPP-SURF, with $T_D = 10$ and $T_C = 0.8$

C) Results as a Function of System Parameters, using CPP-SURF databases

For sake of brevity, the following sections will present a brief analysis of our results. Furthermore, all figures from this point onwards are shown in Appendix D.

We begin by looking at the success rates as a function of the detector threshold (T_D) and the comparator threshold (T_C). The surface plot obtained by matching query images to database objects is displayed in Fig 24, whereas the results obtained by matching query images to database *images* are illustrated in Fig 25. In both figures, we observe that the effects of T_D are barely noticeable (when its value is changed from 10 to 100). A larger T_D will return fewer features, because those with smaller DoH values are pruned away. We originally expected to see worse results when using fewer features, but we then realized that by removing less important features, we were actually reducing the likelihood of incorrectly pairing up dominant features with weak ones.

On the other hand, our results indicate that the value of T_C has a significant impact on the recognition rate. When T_C was set to 0.1, we found that our algorithm terminated without finding any feature pairs at all. This reinforces our theory that a very restrictive comparator will not have any data to work with.

As we increased T_C slightly, the recognition rate immediately peaked, and then began to decrease as we continued to increase T_C. By using a less severe feature pair acceptance criterion, we allowed weaker features to be incorrectly matched against dominant ones. Therefore, we believe that there exists a small range of optimal values (near 0.2) for which only dominant features will be allowed to correctly match to each other, thus achieving the optimal performance.

Interestingly, this linearly decreasing behaviour reverts around values of 0.6 to 0.8, where we began observing better recognition rates once again. We theorize that because the algorithm now allowed almost any feature pairs to contribute to recognition, the weaker features were correctly matching with other weak ones among all images. Because these weak pairs boosted all recognition scores by similar amounts, they in fact cancelled each other out. Therefore, recognition is once again truly based on the dominant features, which explains the high performance rates.

When our system used all available feature pairs ($T_C = 1.0$), we observed that the recognition rate dropped only in the image versus object setup. This is arguably caused by our (not-so-slight) variant where instead of combining features together for each individual object, we combined their feature *pairs* instead. Since it is inevitable that some weak features will be incorrectly paired, their effects accumulated when they were combined in this setup, which reduced the system's performance in the end.

We then restricted our attention to the incorrect object matches, and looked at the position of the correct object in terms of its recognition score position, as illustrated in Fig 26 through Fig 29. Interestingly, we observed that when used values of T_C near 0.7, the average position of the correct match in the image versus image setup was noticeably worse than in the image versus object setup. This observation strengthens our previous argument: when the weaker features were allowed to pair up, the system became incapable of distinguishing between objects in the images. However, this phenomenon is negated by pooling all the bad feature matches and good feature matches together in the image versus object setup.

Finally, Fig 30 through Fig 33 illustrates the number of features available for different parameter settings. This number was not significantly affected by T_D, and grew exponentially with increasing T_C values.

D) Results as a Function of System Parameters, using MAT-SURF databases

After observing very strong results obtained using the original author's implementation of SURF, we were eager to test out our MATLAB version. Since we knew that our SURF framework had some issues and limitations, we restricted our simulations to only using 10 objects instead of 100. Fig 34 through Fig 37 illustrates the recognition rates of our system as a function of T_C and T_D, for different permutations of the detector variant (either SURF-64 or U-SURF) and for different matching schemes (IvI or IvO). We were somewhat disappointed to see that the best recognition rate generated using our oriented system was about 60%. This maximum dropped to about 50% when we omitted feature orientation.

However, we did in fact observe the same effects on the recognition performance as we changed system parameters. Once again, the results were nearly the same for different T_D values, whereas we observed the same three types of responses while tweaking T_C: the system did not have any feature pairs to work with at very low T_C values; then the recognition rate peaked and decreased linearly; and finally the performance improved again as T_C passed through the 0.6 to 0.8 range. Furthermore, we observed that the image versus object setup had less variation because many abnormalities cancelled out when all the feature pairs were grouped by their respective object identity. Finally, because the average correct object position and the average feature count exhibited identically dull results, we decided not to show them.

E) Results as a Function of Database Size, using CPP-SURF databases

Up to this point, our database contained images from all available orientations (i.e. in increments of 5°) because we wanted to use an optimal setup. For this last set of experiments, we decided to assess how many images we can remove from the database while maintaining decent results. Instead of using 72 images per object, we tried databases with 36 images (10° increments), 24 images (15° increments), 12 images (30° increments) and 6 images (60° increments). For these trials, we used CPP-SURF features, a constant T_D value of 10, and T_C values in the range of 0.2 to 0.8, in increments of 0.2.

As shown in Fig 38 and Fig 39, the recognition rate deceased linearly each time we halved the database size. We observed similar behaviour in terms of our other two metrics, and decided not to show them for sake of brevity. Because our results were affected by T_C in the same manner as in our previous experiments, we believe that a value of T_C near 0.8 and an angle increment of about 10° present a healthy balance between database size and system performance.

F) Summary

In the following bullet points, we summarize our findings and suggest some optimal parameter values:

- The detector threshold does not have a significant impact on the recognition results. Even though a smaller T_D will generate many weak results, their effects can be cancelled out by an adequate choice for T_C. Thus, we recommend setting $T_D \approx 10$ for CPP-SURF and $T_D \approx 1$ for MAT-SURF.
- All the effects of the comparator threshold are tightly related to the availability of feature pairs. Therefore, when we are provided with a decent-sized feature database, a T_C value of 0.8 will generate great recognition rates. In fact, SURF's authors also recommended using this value [5].
- Our image versus object variant was able to remove various abnormalities found in our image versus image variant by grouping feature pairs using their respective database object identities.
- Naturally, the system performs best with a populated database. Nevertheless, we believe that using 36 images per object (with an angle increment of 10°) allows for a decent compromise.
- Finally, as we expected, the CPP-SURF system returned the best performance, followed by our oriented version (MAT-SURF64), and then by our non-oriented version (MAT-U-SURF). This is caused by our many design decisions, assumptions, and approximations.

VII. Conclusion

Speeded-Up Robust Features (SURF) is a newly-developed framework, which we believe is very likely to becoming the next *de facto* feature detector in the industry. Compared to its main competition, SIFT, SURF has been shown by its authors to offer both faster and more robust performance. These improvements are made possible by using ingenious box filter and integral image tricks to find features quickly, and by using Haar wavelets to describe them robustly. Using our object recognition task, we have shown how SURF can be used to robustly detect objects in images taken under different extrinsic and intrinsic settings.

For this project, we have developed a full-fledged SURF framework using MATLAB, based solely on the original SURF authors' publications. However, we made some explicit choices in our implementation in some aspects of the algorithm that we did not fully understand, or for which we identified problems in our experiments. Therefore, it is no surprise that our version of SURF yields poorer recognition rates than the original authors' version. Nevertheless, by writing all the algorithms ourselves, we have discovered a tremendous amount of useful information on SURF, as well as on image features in general.

Specifically, our experiments suggest that we should employ a relatively lax comparator threshold, because even though this setting will allow weak features with smaller determinant of Hessian values, our results suggest that their effects cancel out at values near 0.8. However, these large values are better than smaller ones, since we want to have a sufficient number of features to compare with in each image. The same reasoning further suggests choosing a small detector threshold to create more features per image.

Furthermore, our trials have shown that comparing images to objects has the added benefit that the positions of the correct object in erroneous matches often have only slightly smaller recognition scores, which we can use in conjunction with Machine Learning techniques to boost the performance. On the other hand, by matching images to other images, our system can detect both the object identity, as well as the image orientation. We have thus designed two systems, each having their own unique benefits!

We would like to conclude this report by discussing some potential improvements. First of all, because we made many assumptions in our descriptor part, we would like find out which of our assumptions are actually consistent with the original authors' implementation. One simple adjustment is to perform bi-linear interpolation when sampling pixels, instead of choosing the nearest neighbour.

Within our detector, we have disables the fourth octave and did not search through the up-scaled image at all, because our source images had very limited resolutions. We believe that we can learn a lot more about the detector by using larger-sized images. Another related feature that we disabled was the sign of the Laplacian – we are interested to examine the difference in performance resulting from using this feature.

Given time constraints, we did not implement our comparator to truly compare query images with database objects, but instead opted for the simpler approximation of grouping feature pairs together. We suspect that a true IvO setup will improve results in general, but we are curious to find out its side-effects.

Finally, because our comparator codebase does not depend on SURF features, we can trivially use other types of features, such as SIFT, to assess the performance of object matching among different frameworks. Additionally, we can easily adapt our code to accept pixel-based PCA weights, so that we can compare the benefits and drawbacks of using features versus using pixels directly.

In the end we are extremely satisfied with the amount of knowledge that we gained by exploring the various aspects of this powerful feature framework for this project.

Bibliography

1. **Nene, S. A, Nayar, S. K and Murase, H.** CAVE | Software: COIL-100: Columbia Object Image Library. *CS@CU.* [Online] [Cited: 12 15, 2008.] http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php.

2. **Hubel, David H and Wiesel, Torsten N.** Brain Mechanisms of Vision. *Scientific American.* 1979, pp. 150-162.

3. **Lowe, David G.** Object Recognition from Local Scale-Invariant Features. *International Conference on Computer Vision.* 1999.

4. **Bay, Herbert, Tuytelaars, Tinne and Van Gool, Luc.** SURF: Speeded Up Robust Features. *European Conference on Computer Vision.* 2006.

5. **Bay, Herbert, et al.** Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding.* 2008, pp. 346-359.

6. **Lindeberg, Tony.** Feature Detection with Automatic Scale Selection. *International Journal on Computer Vision.* 1998, pp. 79-116.

7. **Harris, Chris and Stephens, Mike.** A Combined Corner and Edge Detector. *ALVEY Vision Conference.* 1988, pp. 147-151.

8. **Kadir, Timor and Brady, Michael.** Scale, Saliency and Image Description. *International Journal on Computer Vision.* 2001, pp. 83-105.

9. **Jurie, Frederique and Schmid, Cordelina.** Scale-Invariant Shape Features for Recognition of Object Categories. *Computer Vision and Pattern Recognition.* 2004, pp. 90-96.

10. **Mindru, Floricia, Tuytelaars, Tenne and Van Gool, Luc.** Moment Invariants for Recognition Under Changing Viewpoint and Illumination. *Computer Vision and Image Understanding.* 2004, pp. 3-27.

11. **Freeman, W.T. and Adelson, E.H.** The Design and Use of Steerable Filters. *Pattern Analysis and Machine Intelligence.* 1991, pp. 891-906.

12. **Carneiro, G and Jepson, A.D.** Multi-Scale Phase-Based Local Features. *Computer Vision and Pattern Recognition.* 2003, pp. 736-743.

13. **Mikolajczyk, Krystian and Schmid, Cordelia.** A Performance Evaluation of Local Descriptors. *Computer Vision and Pattern Recognition.* 2003, pp. 257-263.

14. **Ke, Yan and Sukthankar, Rahul.** PCA-SIFT: A More Distinctive Representation for Local Image Descriptors. *Computer Vision and Pattern Recognition.* 2004, pp. 506-513.

15. **Lowe, David.** Distinctive Image Features from Scale-Invariant keypoints. *International Journal on Computer Vision.* 2004, pp. 91-110.

16. **Grabner, M, Grabner, H and Bischof, H.** Fast Approximated SIFT. *Asian Conference on Computer Vision.* 2006, pp. 918-927.

17. **Beis, Jeffrey S and Lowe, David G.** Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. *Computer Vision and Pattern Recognition.* 1997, pp. 1000-1006.

18. **Omohundro, S.M.** Five Balltree Construction Algorithms. *International Computer Science Institute Technical Report.* 1989, pp. 89-063.

19. **Nistér, D and Stéwenius, H.** Scalable Recognition with a Vocabulary Tree. *Computer Vision and Pattern Recognition.* 2006, pp. 2161-2168.

20. **Datar, M, et al.** Locality-Sensitive Hasing Scheme Based on P-Stable Distributions. *Symposium on Computational Geometry.* 2004, pp. 253-262.

21. **Bay, Herbert.** SURF: Speeded Up Robust Features. [Online] Swiss Federal Institute of Technology Zurich (ETH), 2006. [Cited: 12 15, 2008.] http://www.vision.ee.ethz.ch/~surf/.

22. **Evans, Chris.** OpenSURF: A Simple OpenCV Implementation. [Online] 2005. [Cited: 12 15, 2008.] http://jmkirby.co.uk/.

23. **Marr, D and Hildreth, E.** Theory of Edge Detection. *Royal Society London.* 1980, pp. 187-217.

24. **Neubeck, Alexander and Van Gool, Luc.** Efficient Non-Maximum Suppresion. *International Conference on Pattern Recognition.* 2006.

25. **Brown, Matthew and Lowe, David G.** Invariant Features from Interest Point Groups. *British Machine Vision Conference.* 2002.

26. **Trucco, Emanuele and Verri, Alessandro.** Section 10.4 - Appearance-Based Identification. *Introductory Techniques for 3-D Computer Vision.* New Jersey : Prentice Hall, 1998.

27. **Yong, R.A.** The Gaussian Derivative Model for Spatial Vision: Retinal Mechanisms. *Spatial Vision.* 1987, pp. 273-293.

28. **DeAngelis, G.C, Ohzawa, I and Freeman, R.D.** Receptive-Field Dynamics in the Central Visual Pathways. *Trends in Neuroscience.* 1995, pp. 451-458.

Appendix A: Approximate Nearest Neighbour Algorithm

A) Theory & Implementation

In our feature comparator, even though we did not aim to assess the running speed of our MATLAB implementation, we still looked at the feasibility of finding approximate nearest neighbours. We used a method called Best-Bin-First (BBF) search, proposed by Beis and Lowe, that finds approximate nearest neighbours (ANN) using a K-dimensional (binary) tree structure in constant time [17].

BBF is a type of prioritized depth-first search, with the added heuristic that each internal tree node (i.e. a cut in feature space) is stored in a priority list, sorted by the distance between the query point and the hyper-plane cut. Every time a leaf node is traversed, the algorithm then jumps to the top internal node in the sorted list, and searches the sub-tree in the direction of the unexplored child. The algorithm terminates prematurely after it has travelled to a pre-determined number of leaves, but given that k-d trees are constructed to evenly separate the data with each cut, the approximated nearest neighbour is very likely to be the closest entry.

We designed and implemented three different ways of constructing k-d trees. The most recognized method splits each vector set in the cardinal axis with the largest variance. Once this axis has been determined, the data set is projected onto this hyper-plane, which is achieved by simply looking up the values at the corresponding index. The data set is then split into two subsets, using the median projected value as the dividing threshold.

An alternative construction method is to choose the hyper-plane at each internal node based on the current depth of the tree, or based on a cyclical distribution process. This method ensures that most if not all of the dimensions will have at least one cut. Even though this method is much simpler than the variance-based algorithm, it does not guarantee that the tree structure will maximize the distance between the two subsets of each internal node.

By focusing on the previous criterion of maximizing distance between subsets, we designed a third variant that splits the data along the hyper-plane chosen by the dominant eigenvector of the data covariance matrix. This is implemented using Singular Value Decomposition (SVD). By splitting the data in this manner, we ensure monotonicity with respect to maximum linkage: i.e. the distance between the furthest points in a parent set will always be larger than the distance between furthest points within each of its two children subsets.

The downside of this method is that cuts are no longer guaranteed to be on canonical axes. This means that instead of storing the single value corresponding to a canonical axis index, we need to store a high-dimensional vector representing each hyper-plane cut. Additionally, we must compute Euclidean inner products to find vector projections. However, in theory this variant will guarantee that the closest neighbour for any query point in BBF search can be found with the least amount of tree traversals.

B) Experimental Setup

Rather than testing the effects of using ANN on our entire object recognition system, we decided to scale back our experiments to determine the accuracy of the approximations instead. To achieve this goal, we look for the exact neighbours in the k-d tree based on the BBF strategy, and measure how many leaf traversals are needed before we find the true first two neighbours for each image pair. We base our assessment on the average percentage of traversals required for each type of k-d tree, as well as its range. We decided to conduct two separate data sets, where one set contained different views of the same object, and the other having different objects. By doing so, we wanted to look at whether the performance of ANN changed if we compared similar sets of features versus arbitrary ones. For the first data set we selected 72 images of the first object in the COIL-100 library, whereas for the second data set we randomly chose a single image for each of the 100 distinct objects in the same library. We then conducted our trials on every possible pair-wise permutation, and computed the minimum, average and maximum ratio of traversals required to find the first two exact nearest neighbours. We used the CPP-SURF implementation, and employed a detector threshold of 10, which resulted on average 100 features per image.

C) Experimental results

The resulting metric values for images of the same object are graphed in Fig 11, for the three types of tree construction modes – depth-based, variance-based, and eigenvector-based cuts. We begin by noting that the all three results share the same minimum value, which we subsequently found out that in the best case, all three tree types found the two first true nearest neighbours using only two traversals! We were very surprised to find this phenomenon again when we looked at the results generated for images representing different objects, as shown in Fig 12. We have thus experimentally demonstrated that under certain situations, k-d tree search is capable of attaining the absolute optimal search performance possible!

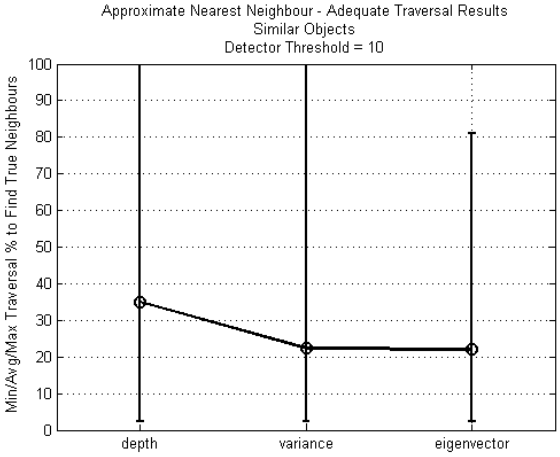

Fig 11: Minimum/average/maximum ratio of traversals over total number of feature points needed to find the two exact nearest neighbours for images of the same object

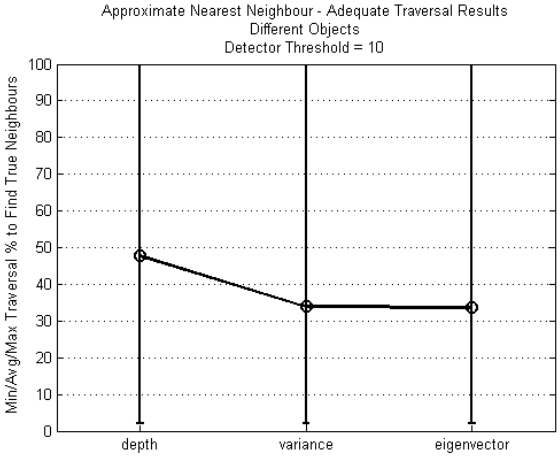

Fig 12: Minimum/average/maximum ratio of traversals over total number of feature points needed to find the two exact nearest neighbours for images of different objects

However, because not every case can be so optimal, we then looked at the mean traversal percentage to compare the performance of the three types of trees in the average case. For both types of objects used, the average ratio of traversal clearly demonstrates the wide performance gap between the depth-based construction mode and the two other modes. Since we suspected that the quality of BBF search depended on the criterion of maximizing (or nearly maximizing) the distance between the children of each hyper-plane cut, we were happy to find experimental confirmation within these results.

When we looked at the actual numerical values of the average, we found that the eigenvector mode always outperformed the variance mode, but their difference is very negligible. Considering that we needed significantly more space and time to process eigenvector-based trees, we believe that using variance to determine cuts will result in a very favourable compromise between resource requirements and performance accuracy on average.

The maximum number of traversals demonstrated that under almost all settings, the entire k-d tree need to be search to find the true neighbours in the worst case. The only exception is the result obtained for images of the same object, with eigenvector-based trees. However, this setting is not representative of a real-life setup, because true databases will always contain features originating from different objects. Therefore, we must be wary of its horrible worst case performance when using BBF and k-d trees to find approximate nearest neighbours.

Finally, we re-iterate the fact that results for images of different objects share a very similar structure to those for images of the same object. The average number of traversals needed in particular differs only by an offset among the two types of images used.

D) Conclusion

We have determined that the approximate nearest neighbour concept based on K-Dimensional trees and the Best-Bin-First strategy has many desirable properties. For example, in the best case, the true neighbours are found immediately; whereas in the average case of a real-life setup (i.e. using a database with features coming from different objects), it is possible to find the true first two neighbours after traversing approximately 35% of all features. This last result requires the k-d tree to be constructed intelligently, using either data variance or the principal component to determine hyper-plane cuts through the feature space. We believe that among these two modes, using the data variance strikes a favourable balance between requirements and performance.

Appendix B: Supplementary Project Information

A) Online Resources

I. Entire MATLAB Codebase and .MAT Trial Results:
www.cim.mcgill.ca/~anqixu/Courses/COMP558/SURF.Codebase.zip

II. Graph of Results for Individual Trials:
www.cim.mcgill.ca/~anqixu/Courses/COMP558/SURF.Figures.zip

III. Original C++ SURF Implementation, for Linux, v1.0.9:
http://www.vision.ee.ethz.ch/~surf/SURF-V1.0.9.tar.gz

IV. COIL-100 PNG Image Library:
http://www.cs.columbia.edu/CAVE/databases/SLAM_coil-20_coil-100/coil-100/coil-100.zip

B) How to Generate Feature Database using C++-SURF and COIL-100

1. Download and extract I, III, IV into separate folders
2. Execute the following in a Linux terminal:
   ```
   > <MAT-SURF Folder>/CPP_SURF_SCRIPTS/processCOILdb <CPP-SURF Folder>/surf.ln
   >   <Detector Threshold> <COIL-100 Folder> <Destination Folder>
   ```
3. Edit line 15 of `<MAT-SURF Folder>/Global/LOAD_SURF_DATA` to point the variable `path` to `<Destination Folder>`
4. Execute `<MAT-SURF Folder>/Global/LOAD_SURF_DATA` in MATLAB
5. (OPTIONAL) Execute `save(<MAT DB Filename>, 'data');` in MATLAB to save database

C) How to Generate Feature Database using MAT-SURF and COIL-10 Subset

1. Download and extract I & IV into separate folders
2. Modify `<MAT-SURF Folder>/Global/GENERATE_COIL10_DATABASES` to set the correct path to the COIL-100 library, the parameters for each database, and their database.MAT filenames
3. Execute `<MAT-SURF Folder>/Global/GENERATE_COIL10_DATABASES` in MATLAB

D) How to Generate Features using C++-SURF for a Single Image

1. Download and extract I, III, IV into separate folders
2. Execute the following in a Linux terminal:
   ```
   > <MAT-SURF Folder>/CPP_SURF_SCRIPTS/processSURFimg <CPP-SURF Folder>/surf.ln
   >   <Detector Threshold> <Destination Folder>/result.surfmat
   ```
3. Execute `result = load('<Destination Folder>/result.surfmat');` in MATLAB

E) How to Execute Trials Related to T_D and T_C Parameters

1. Download and extract I into a folder
2. Create appropriate database(s)
3. Modify `<MAT-SURF Folder>/Global/TEST_CT_DT` to select the specific objects (`OBJ_IDs`), angle IDs for the training database (`T_ANGLE_IDs`), angle IDs for the query set (`Q_ANGLE_IDs`), the comparator threshold range (`ctRange`), the database.MAT file (`dbFile`), the detector threshold(s) (`dt`) and the filename containing results (`saveFile`)
4. Execute `<MAT-SURF Folder>/Global/TEST_CT_DT` in MATLAB

F) How to Execute Angle-Increment Trials

1. Download and extract I into a folder
2. Create appropriate database(s)
3. Edit line 2 of `<MAT-SURF Folder>/Global/TEST_CPP_CT_ANGLES` to point the variable `dbFile` to the appropriate database(s)
4. Execute `<MAT-SURF Folder>/Global/TEST_CPP_CT_ANGLES` in MATLAB

G) How to Execute Approximate Nearest Neighbour Trials

1. Download and extract I into a folder
2. Create appropriate database(s)
3. Edit line 4 of `<MAT-SURF Folder>/Global/TEST_ANN_TREE_VS_DIRECT` to point the variable `dbFile` to the appropriate database
4. Execute `<MAT-SURF Folder>/Global/TEST_ANN_TREE_VS_DIRECT` in MATLAB

H) How to Generate Figures & Summary Data for Results Related to T_D and T_C Parameters

1. Download and extract I into a folder
2. Execute `<MAT-SURF Folder>/Global/PROCESS_CT_DT_RESULTS` in MATLAB

I) How to Generate Figures & Summary Data for Angle-Increment Results

1. Download and extract I into a folder
2. Execute `<MAT-SURF Folder>/Global/PROCESS_ANGLE_INCR_RESULTS` in MATLAB

J) How to Generate Figures & Summary Data for Approximate Nearest Neighbour Results

1. Download and extract I into a folder
2. Execute `<MAT-SURF Folder>/Global/PROCESS_TREE_RESULTS` in MATLAB

Appendix C: Sample Images in the COIL-100 Library

Please note that all of the images below are borrowed without modifications and without permission from the Columbia Object Image Library (COIL-100), which is provided freely by Columbia University [1].

Fig 13: Collective photo of all 100 objects in the COIL-100 library

Our hand-picked COIL-10 subset contains the following objects:

Fig 14: Object 0

Fig 15: Object 3

Fig 16: Object 5

Fig 17: Object 7

Fig 18: Object 8

Fig 19: Object 14

Fig 20: Object 28

Fig 21: Object 30

Fig 22: Object 33

Fig 23: Object 44

Appendix D: Experiment Result Graphs

A) Results as a Function of System Parameters, using CPP-SURF databases

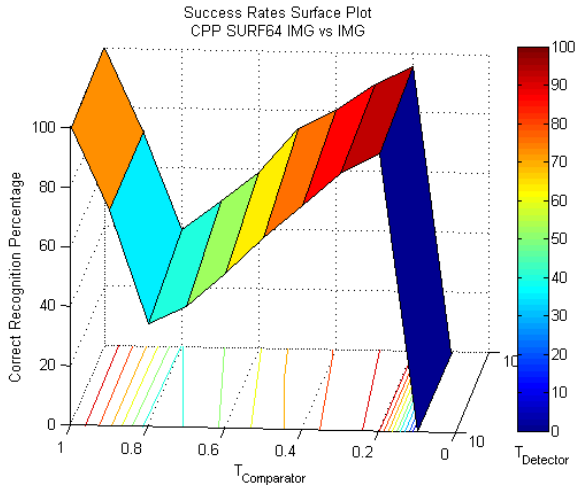

Fig 24: Success rates as a function of T_D and T_C, for CPP-SURF, and for database image matches (IvI)

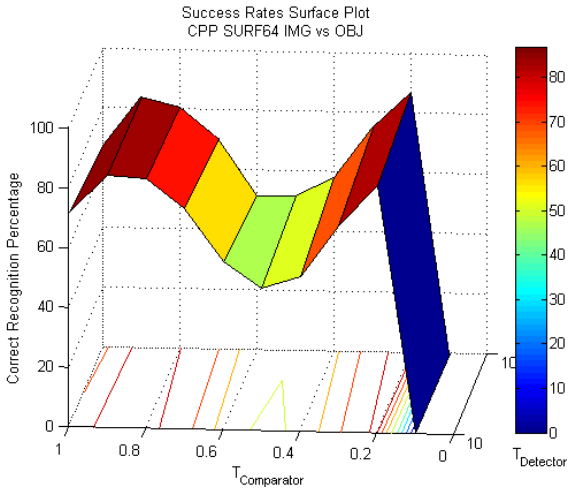

Fig 25: Success rates as a function of T_D and T_C, for CPP-SURF, and for database object matches (IvO)

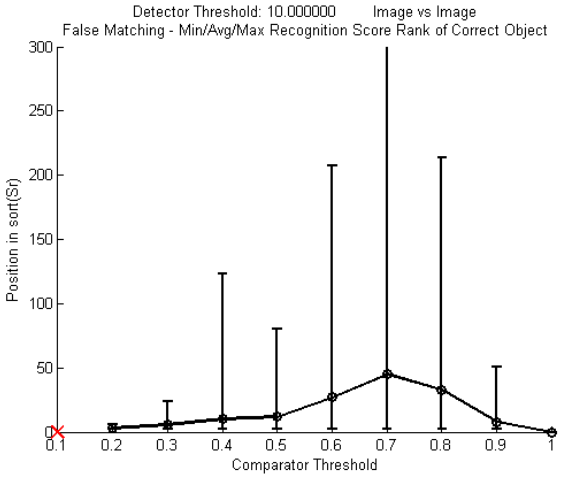

Fig 26: Correct object's S_R position in erroneous results as a function of T_C, for CPP-SURF and IvI; T_D = 10

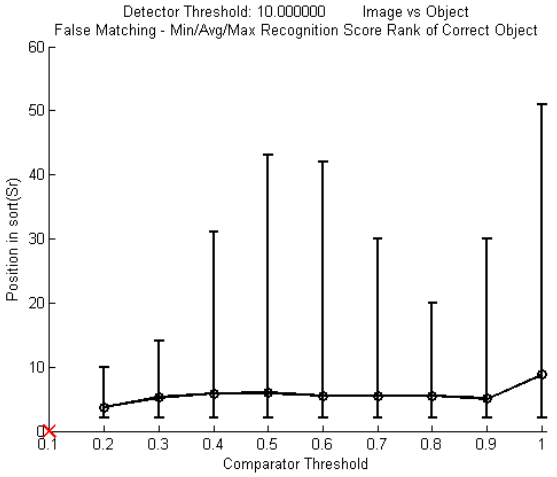

Fig 27: Correct object's S_R position in erroneous results as a function of T_C, for CPP-SURF and IvO; T_D = 10

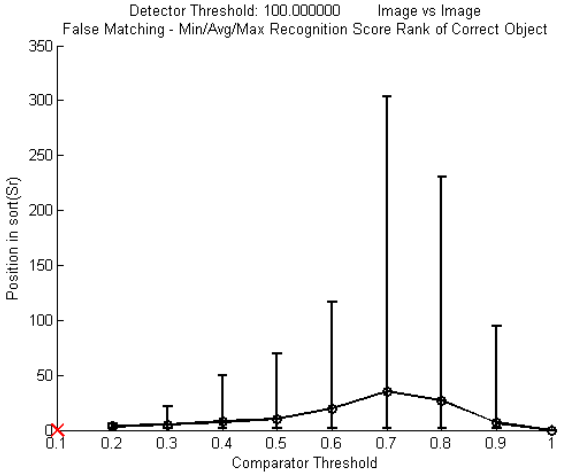

Fig 28: Correct object's S_R position in erroneous results as a function of T_C, for CPP-SURF and IvI; T_D = 100

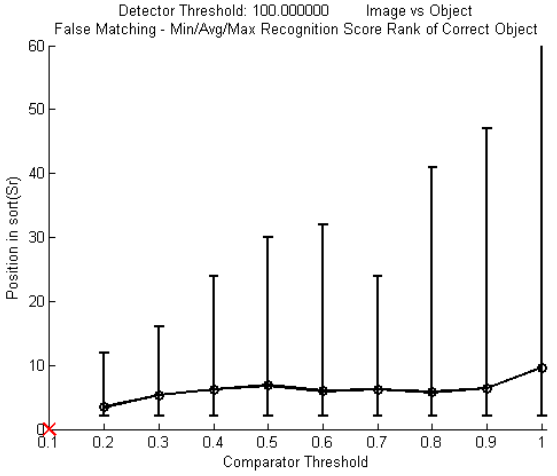

Fig 29: Correct object's S_R position in erroneous results as a function of T_C, for CPP-SURF and IvO; T_D = 100

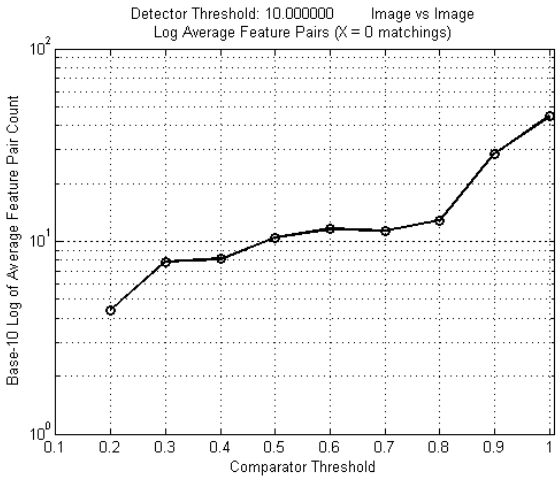

Fig 30: Log_{10} for average number of feature pairs as a function of T_C, for CPP-SURF and IvI; $T_D = 10$

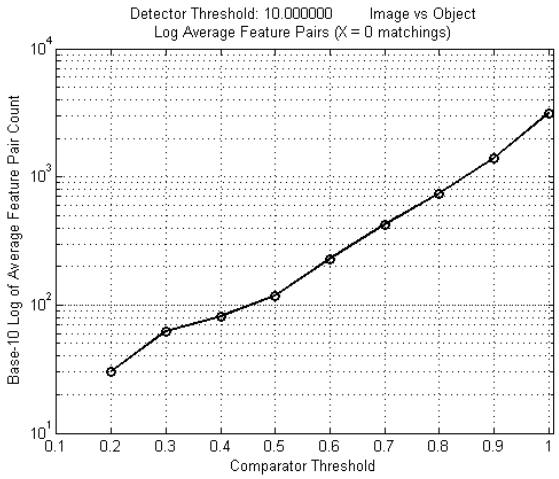

Fig 31: Log_{10} for average number of feature pairs as a function of T_C, for CPP-SURF and IvO; $T_D = 10$

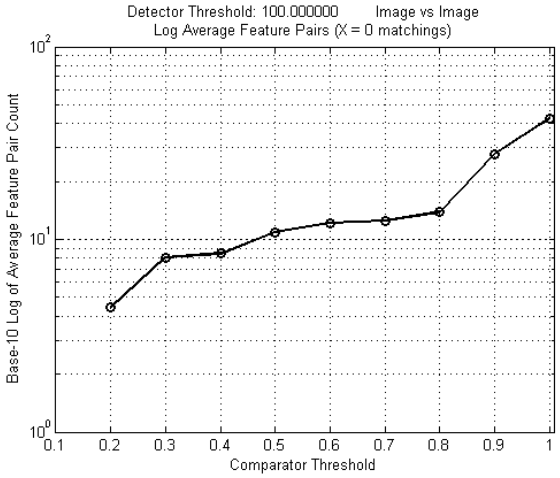

Fig 32: Log_{10} for average number of feature pairs as a function of T_C, for CPP-SURF and IvI; $T_D = 100$

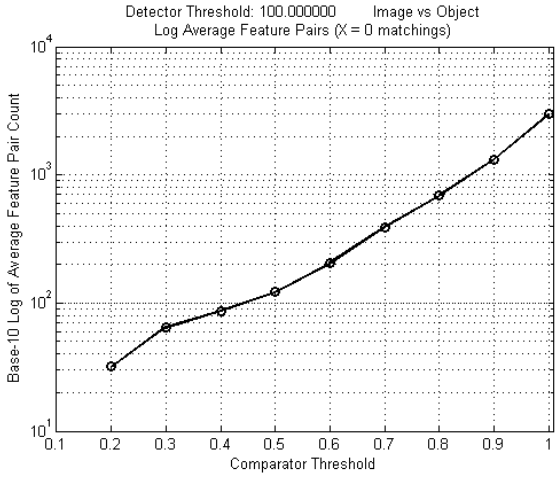

Fig 33: Log_{10} for average number of feature pairs as a function of T_C, for CPP-SURF and IvO; $T_D = 100$

B) Results as a Function of System Parameters, using MAT-SURF databases

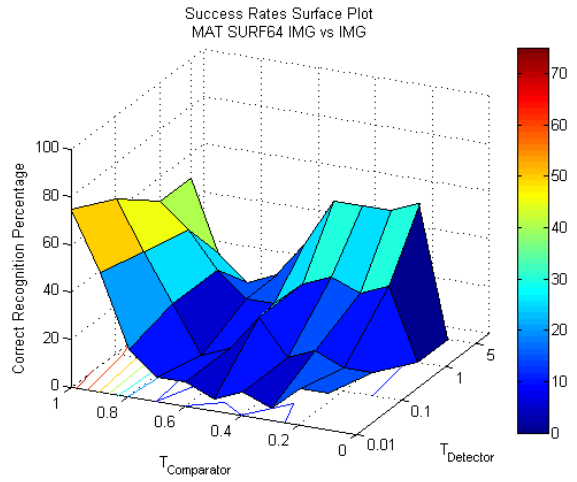

Fig 34: Success rates as a function of T_D and T_C, for MAT-SURF64, and for database image matches (IvI)

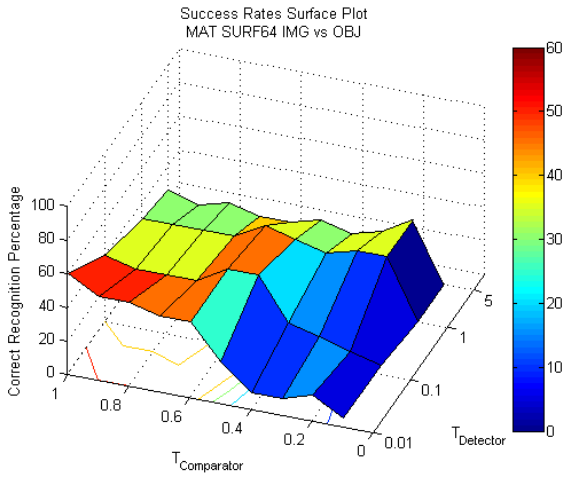

Fig 35: Success rates as a function of T_D and T_C, for MAT-SURF64, and for database object matches (IvO)

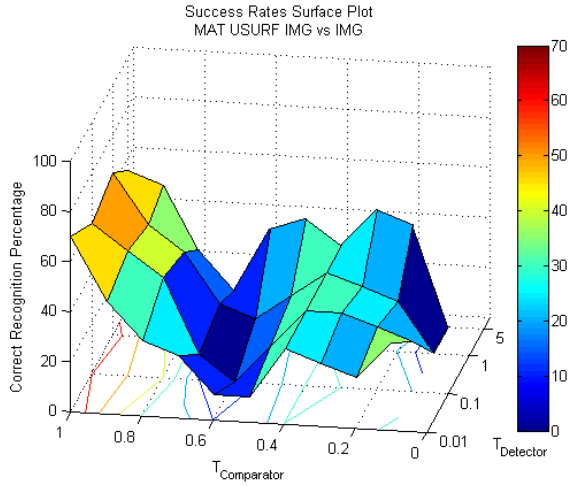

Fig 36: Success rates as a function of T_D and T_C, for MAT-U-SURF, and for database image matches (IvI)

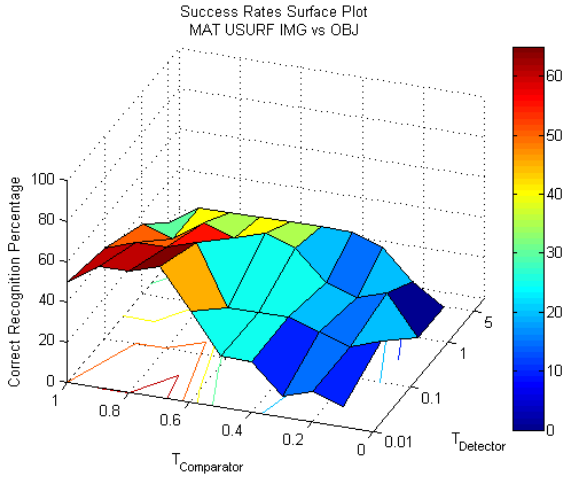

Fig 37: Success rates as a function of T_D and T_C, for MAT-U-SURF, and for database object matches (IvO)

C) Results as a Function of Database Size, using CPP-SURF databases

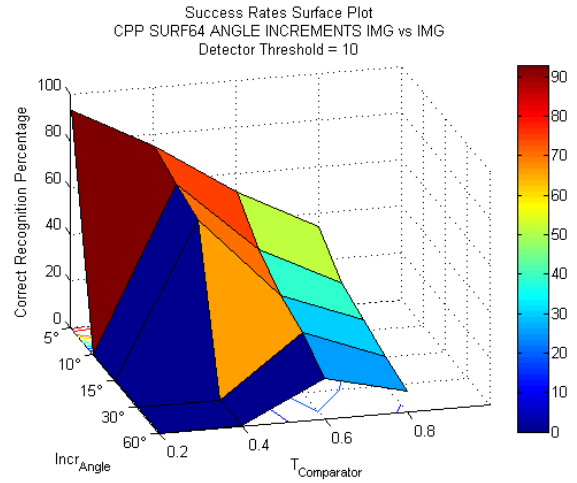

Fig 38: Success rates as a function of I_{ANGLE} and T_C, for CPP-SURF, and for database image matches (IvI)

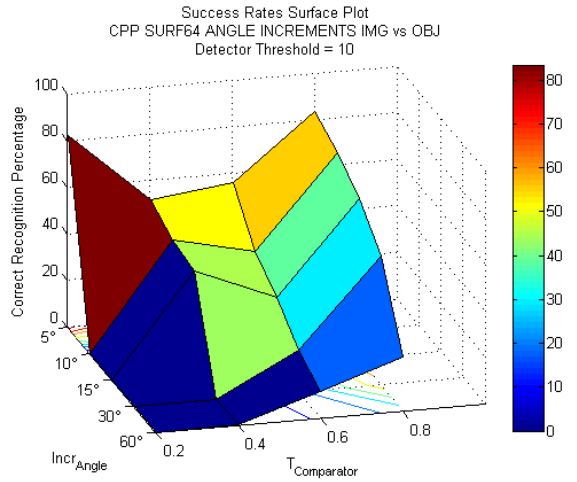

Fig 39: Success rates as a function of I_{ANGLE} and T_C, for CPP-SURF, and for database object matches (IvO)