**Overview**:

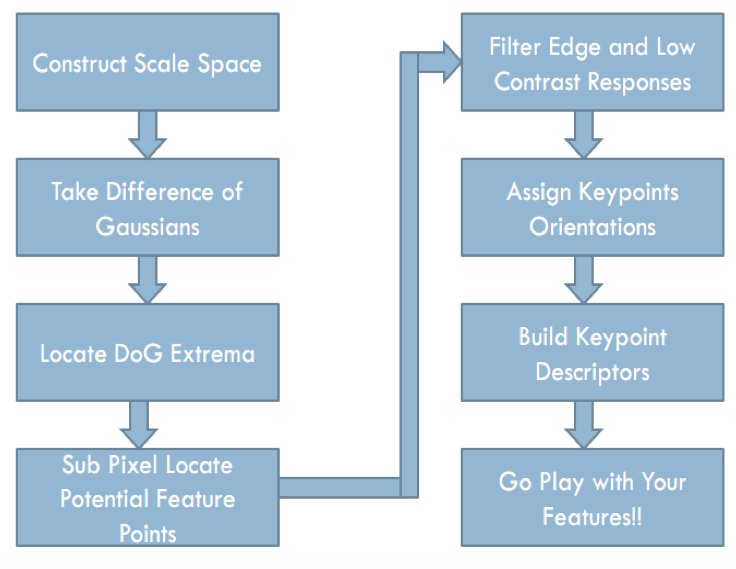
This project is a comparative study of different Image tampering detection algorithms.

The comparative studies assess the performance of the algorithms based on the results obtained in different criteria such as number of feature points, feature descriptors and others. These studies are an important resource to understand the behavior of the algorithms and their influence on the results obtained. Mainly we are concentrating on algorithms SIFT, SURF and FAST.

**Detailed Study**:

SIFT ALGORITHM: Here's an outline of what happens in SIFT.

1. **Constructing a Scale space:** This is the initial preparation. You create internal representations of the original image to ensure scale invariance. Generating a “scale space” does this.
2. **LOG Approximation:** The Laplacian of Gaussian is great for finding interesting points (or key points) in an image. But it's computationally expensive. So we cheat and approximate it using the representation created earlier.
3. **Finding Key points**: With the super fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in step 2
4. **Removal of bad key points**: Edges and low contrast regions are bad key points. Eliminating these makes the algorithm efficient and robust. A technique similar of Harris corner detector used here.
5. **Assigning an orientation to the key points:** An orientation is calculated for each key point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.
6. **Generate SIFT Features:** Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features.



SURF ALGORITHM:

The SURF algorithm is very similar to the SIFT algorithm when it comes to feature detection. The main difference relies on the use of an integral image from the first as the basis for this detection. The use of a very basic Hessian-matrix approximation facilitates the implementation of the integral image concept, which in turn speeds up the detection. The detector is based on the Hessian matrix because of its good performance in accuracy. More precisely, the algorithm detects blob-like structures at locations where the determinant of this matrix is maximum.

One important reason to implement this feature extraction method is because it provides complementary information about the region of interest that cannot be obtained from edge or corner detectors. Also, the improvement on speed achieved with this implementation is a very desirable factor for many processes

FAST ALGORITHM:

FAST is a feature-based algorithm. Because of this difference the process through which FAST detects feature points varies significantly from SIFT and SURF. FAST relies on a corner response function (CRF) to robustly detect corners in a given scene. It also used a multi grid algorithm to detect corners that speeds the process significantly. The three-step multi grid algorithm used to detect comers is presented

Below:

Step 1:In a low-resolution image; compute the simple CRF at every pixel location. Classify pixels with a response higher than a define threshold T1 as potential corners‟.

Step 2:Using the full resolution image for each potential corner pixel, compute the CRF. If the response is lower than another threshold already detected, then the pixel is not a corner, and the upcoming step is not performed. If not, use a inter pixel approximation and compute a new response. If the response is lower than the second threshold T2 then the pixel is not a comer.

Step 3: Find pixels with a locally maximal CRF and mark them as corners. This step is necessary since in the vicinity of a corner more than one point will have high CRF, and only the largest CRF is declared to be a comer point. This is called non-maximum suppression (NMS).

OBSERVATIONS:

While implementing the feature detection component on the images it was found that despite the image, SIFT detects more features than FAST or SURF.

SURF follows SIFT in the amount of features detected but the difference is considerable between both for the majority of the images. SURF, although finding less, detected more “robust” features (i.e. features with enough information for later match).

FAST on the other hand found very few features in all of the images tested. Given FAST is a feature-based detector; one could expect a higher amount of detections. But corner detectors usually limit themselves to detect pixels in the image with high contrast and rejected any other. Although features with more information are more desirable, this seemed to harm the performance of FAST in the quantitative aspect, but the quality of the features is undoubtedly among the best.

**Code**:

% Image tampering detection using SIFT algorithm and Discrete Wavelet transform

% Using VL feat Library for SIFT algorithm

Original\_Image = imread('Original.jpg');

Tampered\_Image = imread('Tampered.jpg');

Original\_Image = rgb2gray(Original\_Image);

Tampered\_Image = rgb2gray(Tampered\_Image);

————— For time being don’t touch it ——————————————

% Extracting the red component of the image and performing DWT on it.

% Original\_Image = im2double(Original\_Image);

% redcomp = Original\_Image(:,:,3);

% [A H V D]=dwt2(redcomp, ‘haar');

% New\_I = wcodemat (A);

% Image (new\_I);

——————————————————————————————————————————

SIFT using vl\_libraray

% The vl\_sift command requires a single precision gray scale image.

New\_Original\_Image = single (rgb2gray (Original\_Image));

Image (new\_Original\_Image);

New\_Tampered\_Image = single (rgb2gray (Tampered\_Image));

Image (new\_Tampered\_Image);

% f gives the frames and d gives the descriptors of the sift algorithm

[OriginalFrames, OriginalDescriptors] = vl\_sift (new\_Original\_Image);

[TamperedFrames, TamperedDescriptors] = vl\_sift (new\_Tampered\_Image);

% Where 1.5 = ratio between Euclidean distance of NN2/NN1

[Matches score] = vl\_ubcmatch (OriginalDescriptors, TamperedDescriptors, 1.5);

Subplot (1,2,1);

Imshow (uint8 (Original\_Image));

Hold on;

Plot (OriginalFrames (1,matches (1,:)), OriginalFrames (2,matches (1,:)),'b\*');

Subplot (1,2,2);

Imshow (uint8 (Tampered\_Image));

Hold on;

Plot (TamperedFrames (1,matches (2,:)), TamperedFrames (2,matches (2,:)),'r\*');

—————— SURF ———————————

Original\_Surf\_pts = detectSURFFeatures (Original\_Image);

Original\_Surf\_pts =

147x1 SURF Points array with properties:

Scale: [147x1 single]

SignOfLaplacian: [147x1 int8]

Orientation: [147x1 single]

Location: [147x2 single]

Metric: [147x1 single]

Count: 147

Tampered\_Surf\_pts = detectSURFFeatures (Tampered\_Image);

158x1 SURF Points array with properties:

Scale: [158x1 single]

SignOfLaplacian: [158x1 int8]

Orientation: [158x1 single]

Location: [158x2 single]

Metric: [158x1 single]

Count: 158

[Original Features, Original Points] = extract Features (Original\_Image, Original\_Surf\_pts);

[Tampered Features, Tampered Points] = extract Features (Tampered\_Image, Tampered\_Surf\_pts);

Index Pairs = match Features (Original Features, Tampered Features);

Matched Original = Original Points (index Pairs (:, 1));

Matched Tampered = Tampered Points (index Pairs (:, 2));

Figure;

ShowMatchedFeatures (Original\_Image, Tampered\_Image, matched Original, matched Tampered);

—————— FAST ——————————

Original\_Surf\_pts = detectFASTFeatures (Original\_Image);

Original\_Surf\_pts =

239x1 corner Points array with properties:

Location: [239x2 single]

Metric: [239x1 single]

Count: 239

Tampered\_Surf\_pts = detectFASTFeatures (Tampered\_Image);

Tampered\_Surf\_pts =

240x1 corner Points array with properties:

Location: [240x2 single]

Metric: [240x1 single]

Count: 240

[Original Features, Original Points] = extract Features (Original\_Image, Original\_Surf\_pts);

[Tampered Features, Tampered Points] = extract Features (Tampered\_Image, Tampered\_Surf\_pts);

Index Pairs = match Features (Original Features, Tampered Features);

Matched Original = Original Points (index Pairs (:, 1));

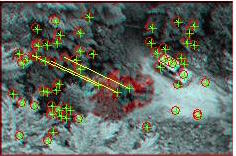
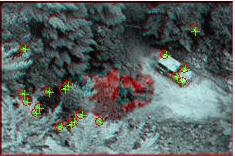
Matched Tampered = Tampered Points (index Pairs (:, 2));

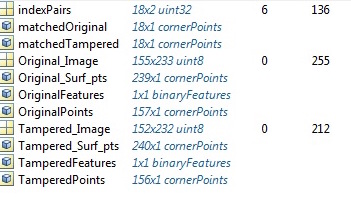
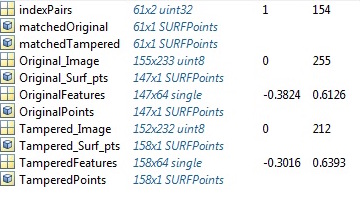
Figure;

ShowMatchedFeatures (Original\_Image, Tampered\_Image, matched Original, matched Tampered);

**Results:**

**** Original Image **** Tampered

**** SURF Results  FAST



**Conclusion**:

There is no standard algorithm that can detect all types of Image tampering’s. Every algorithm has it’s own advantages and disadvantages. Out of all algorithms we considered, SIFT algorithm is more robust in finding feature points but it is a slower than remaining two algorithms. SURF algorithm is less robust and faster compared to SIFT. FAST algorithm is faster compared to other 2 algorithms but it is very unreliable in Image tampering detection.

**References**:

<http://www.cs.dartmouth.edu/farid/>

<http://blog.jgc.org/2008/02/tonight-im-going-to-write-myself-aston.html>

<http://reference.wolfram.com/language/ref/ImageCorrespondingPoints.html>

<http://naveenshanmugam.blogspot.com/2014/02/forgery-image-detection-image.html>

<http://www.mathworks.com/matlabcentral/fileexchange/45187-a-simple-sift-implementation-with-pose-estimation/content//imagematch.m>

<http://www.vlfeat.org/>

<http://www.vlfeat.org/overview/sift.html>

<http://www.mathworks.com/help/images/examples/find-image-rotation-and-scale-using-automated-feature-matching.html>

<http://www.mathworks.com/help/vision/feature-detection-extraction-and-matching.html>