Feature, Keypoints and Descriptors
Feature Matching

ÁP DỤNG

- Nhận dạng (object recognition)
- Bám vật
- Thị giác 3D
- Camera calibration
- Ghép ảnh panorama (stitching)
- Tìm ảnh (image retrival) , đăng ký ảnh (image registration)
- Định vị robot…

Corner Finding

Harris-Shi-Tomasi feature detector
Simple blob detector
SIFT feature detector

SURF feature detector

Star/CenSurE feature detector

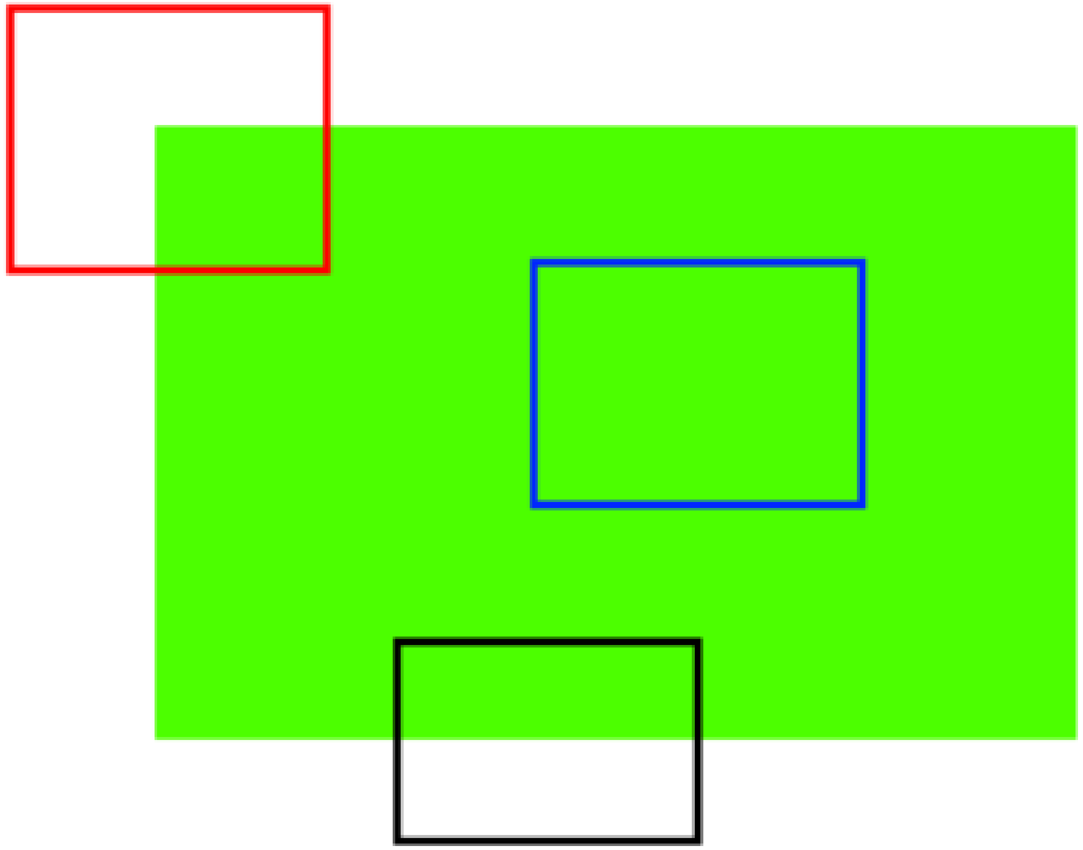
Optical Flow
Keypoint Filtering
Matching Methods
Displaying Results

TÌM ĐIỂM QUAN TRỌNG

- Tìm đặc trưng (feature detection) là tìm các điểm đặc trưng cho một vật ảnh)
- Đối sánh đặc trưng (feature matching) là quá trình so sánh các điểm đặc trưng giữa hai vật để tìm sự giống nhau giữa hai vật
- Các đặc trưng cần biểu diễn cho vật dưới các sự biến đổi hình học quay, dời , tỷ lệ, bị che khuất, ảnh sáng thay đổi…
- Các đặc trưng bền vững là các góc, ngã ba, đốm, cạnh, gọi là điểm quan trọng (Interest Points. Key Points)

E F: Key point

How do we find the corners?. look for the regions in images which have maximum variation when moved (by a small amount) in all regions around it. Finding these image features is called Feature Detection. Computer also should describe the region around the feature so that it can find it in other images. So called description is called Feature Description



Harris Corner Detection

- Thuật toán tìm góc
- Đầu tiên tính sự thay đổi cường độ sáng khi di chuyển cửa sổ một giá trị nhỏ

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

$$E(u,v) \approx [u \;\; v] \, M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

I_x and I_y are image derivatives in x and y directions respectively.

Harris Corner Detection

- Tìm xem cửa sổ có thể chứa điểm góc hay không

$$R = \det(M) - k(\mathrm{trace}(M))^2$$

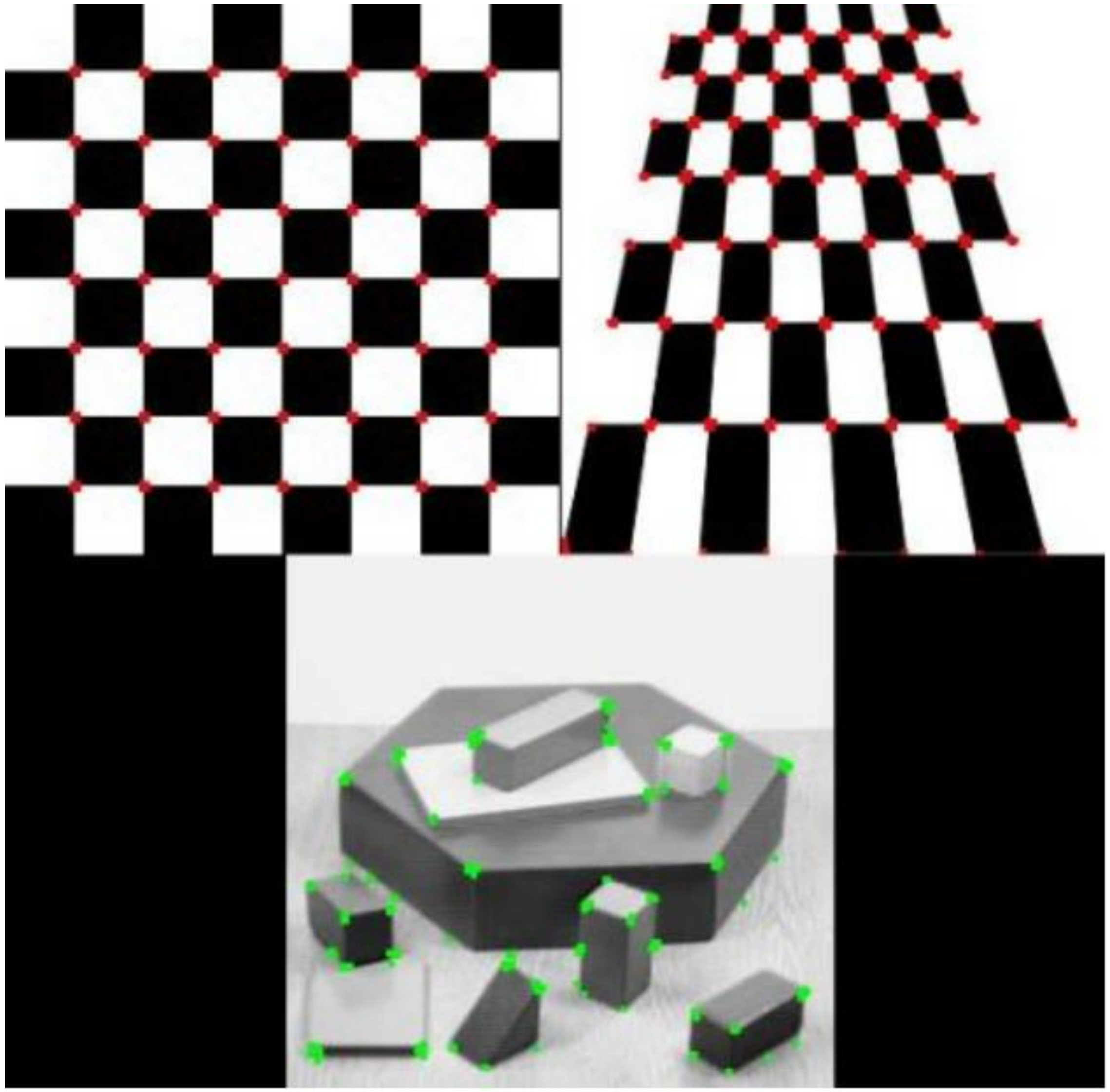
$$\det(M) = \lambda_1 \lambda_2$$
$$\mathrm{trace}(M) = \lambda_1 + \lambda_2$$
λ_1 and λ_2 are the eigen values of M

$|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
$R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

Harris Corner Detection

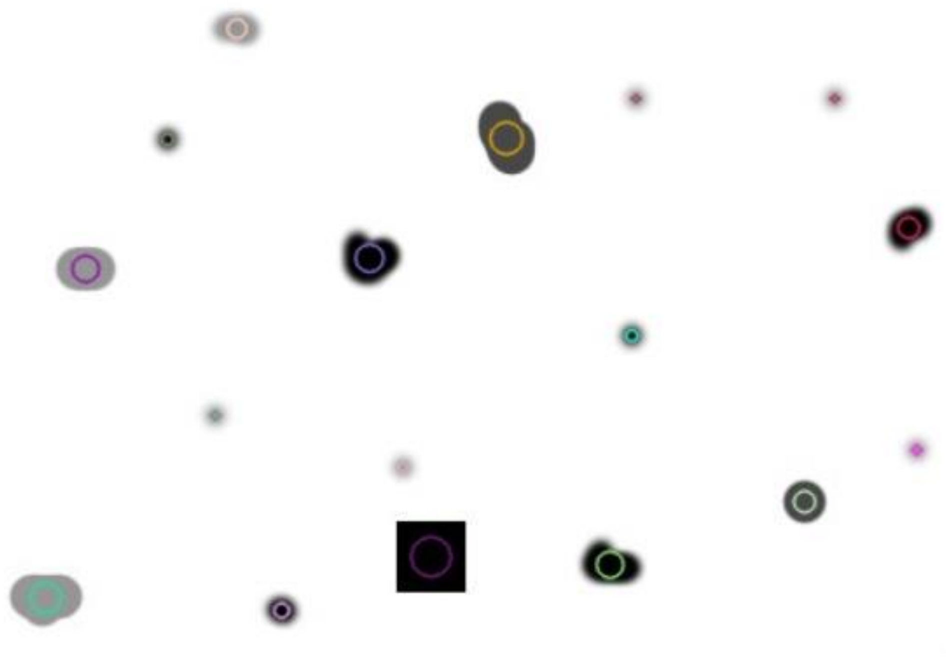
```
import cv2
import numpy as np
filename = 'chessboard.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the
image.
img[dst>0.01*dst.max()]=[0,0,255]
cv2.imshow('dst',img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Harris Corner Detection



Blob Detection

- A Blob is a group of connected pixels in an image that share some common property (E.g grayscale value). The goal of blob detection is to identify and mark these blobs.



cv::SimpleBlobDetector

- SimpleBlobDetector convert gray image to binary and detect black blob on white background or white blob on black background . The return value is location of center and size of blob.

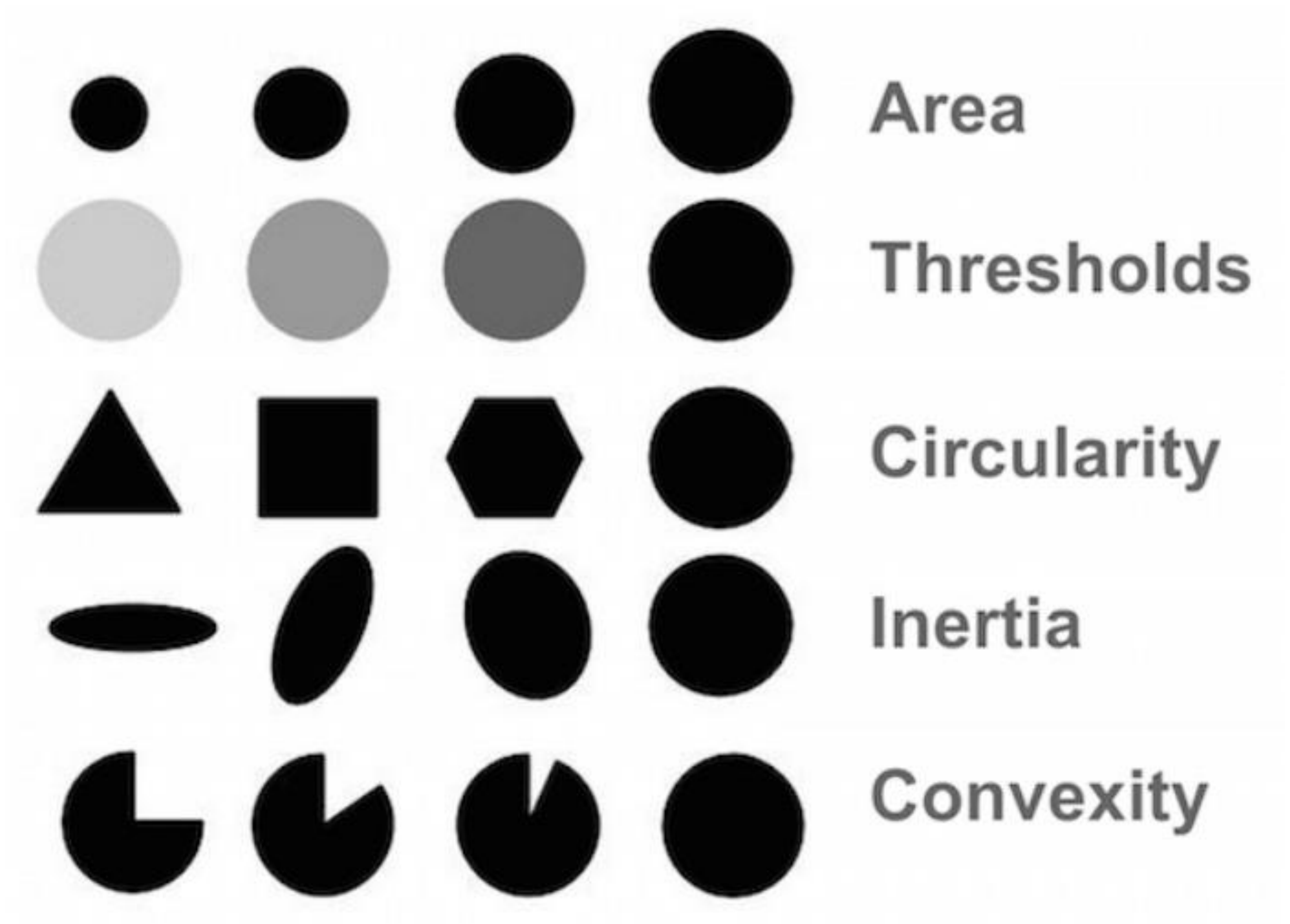
This class performs several filtrations of returned blobs. You should set filterBy* to true/false to turn on/off corresponding filtration. Available filtrations:

By color. This filter compares the intensity of a binary image at the center of a blob to blobColor. If they differ, the blob is filtered out. Use blobColor = 0 to extract dark blobs and blobColor = 255 to extract light blobs.

By area. Extracted blobs have an area between minArea (inclusive) and maxArea (exclusive).

By circularity. Extracted blobs have circularity ({4*pi*Area}/{perimeter ^2}) between minCircularity (inclusive) and maxCircularity (exclusive).

SimpleBlobDetector



By ratio of the minimum inertia to maximum inertia. Extracted blobs have this ratio between minInertiaRatio (inclusive) and maxInertiaRatio (exclusive).

By convexity. Extracted blobs have convexity (area / area of blob convex hull) between minConvexity (inclusive) and maxConvexity (exclusive).

Default values of parameters are tuned to extract dark circular blobs.

```
// Setup SimpleBlobDetector parameters.
SimpleBlobDetector::Params params;
 // Change thresholds
params.minThreshold = 10;
params.maxThreshold = 200;
```

```
// Filter by Area.
params.filterByArea = true;
params.minArea = 1500;
// Filter by Circularity
params.filterByCircularity = true;
params.minCircularity = 0.1;
 // Filter by Convexity
params.filterByConvexity = true;
params.minConvexity = 0.87;
 // Filter by Inertia
params.filterByInertia = true;
params.minInertiaRatio = 0.01;
```

SimpleBlobDetector Python

```
# Standard imports
import cv2
import numpy as np;
 # Read image
im = cv2.imread("blob.jpg", cv2.IMREAD_GRAYSCALE)
 # Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector()
 # Detect blobs.
keypoints = detector.detect(im)
# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the
size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]),
(0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
 # Show keypoints
cv2.imshow("Keypoints", im_with_keypoints)
cv2.waitKey(0)
```

SimpleBlobDetector C++

```
using namespace cv;
// Read image
Mat im = imread( "blob.jpg", IMREAD_GRAYSCALE );
 // Set up the detector with default parameters.
SimpleBlobDetector detector;
 // Detect blobs.
std::vector<KeyPoint> keypoints;
detector.detect( im, keypoints);
 // Draw detected blobs as red circles.
// DrawMatchesFlags::DRAW_RICH_KEYPOINTS flag
ensures the size of the circle corresponds to the size of
blob
Mat im_with_keypoints;
```

SimpleBlobDetector

```
drawKeypoints( im, keypoints, im_with_keypoints,
Scalar(0,0,255),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
 // Show blobs
imshow("keypoints", im_with_keypoints );
waitKey(0);
// Setup SimpleBlobDetector parameters.
SimpleBlobDetector::Params params;
 // Change thresholds
params.minThreshold = 10;
params.maxThreshold = 200;
 // Filter by Area.
params.filterByArea = true;
```

SimpleBlobDetector

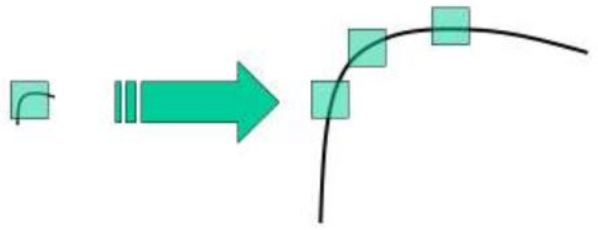
```
params.minArea = 1500;
 // Filter by Circularity
params.filterByCircularity = true;
params.minCircularity = 0.1;
 // Filter by Convexity
params.filterByConvexity = true;
params.minConvexity = 0.87;
 // Filter by Inertia
params.filterByInertia = true;
params.minInertiaRatio = 0.01;
 #if CV_MAJOR_VERSION < 3   // If you are using
OpenCV 2
```

SimpleBlobDetector

```
// Set up detector with params
  SimpleBlobDetector detector(params);
 // You can use the detector this way
  // detector.detect( im, keypoints);
 #else
   // Set up detector with params
  Ptr<SimpleBlobDetector> detector =
SimpleBlobDetector::create(params);
   // SimpleBlobDetector::create creates a smart pointer.
  // So you need to use arrow ( ->) instead of dot ( . )
  // detector->detect( im, keypoints);
 #endif
```

SIFT (Scale-Invariant Feature Transform)

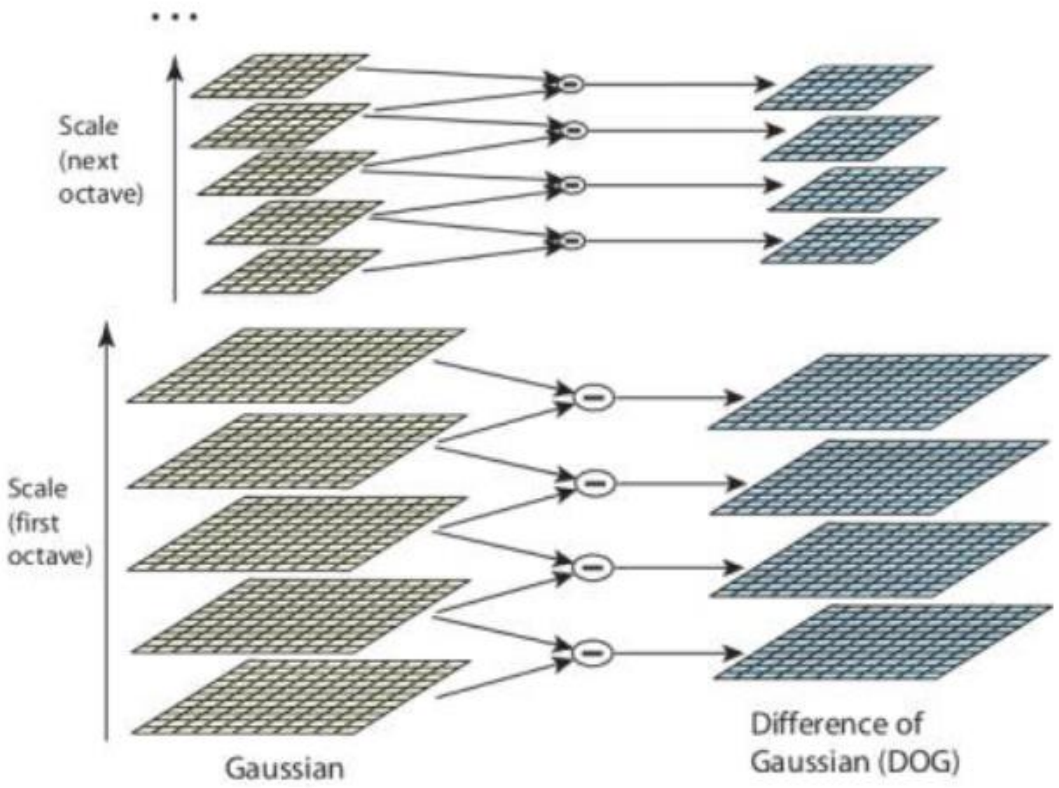
- Harris corner is rotation-invariant but not scale invariant



- We can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scale-space filtering is used. Laplacian of Gaussian is found for the image with various sigma values. LoG acts as a blob detector which detects blobs in various sizes due to change in sigma, sigma acts as a scaling parameter. gaussian kernel with low sigma gives high value for small corner while guassian kernel with high sigma fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x,y,sigma) values which means there is a potential keypoint at (x,y) at sigma scale.

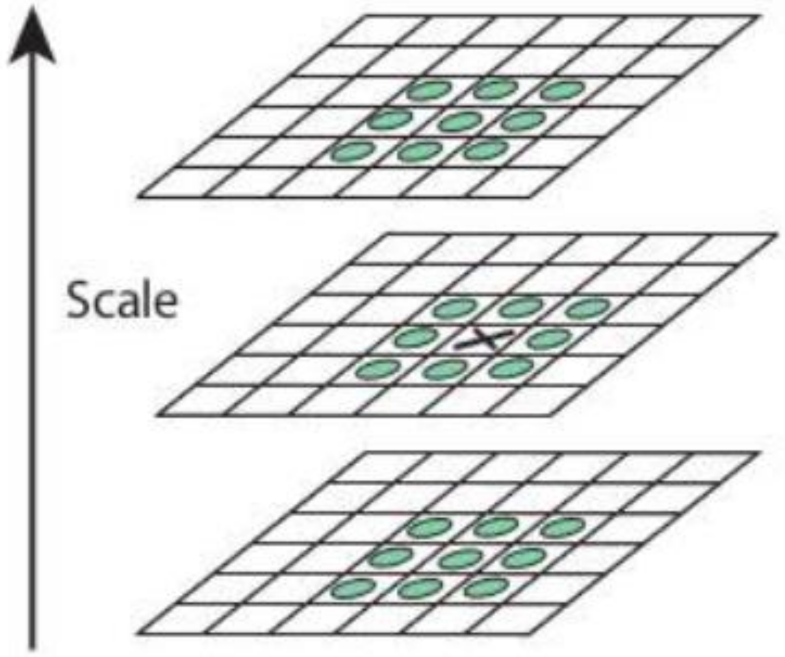
SIFT (Scale-Invariant Feature Transform)

- SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different sigma, let it be sigma and ksigma. This process is done for different octaves of the image in Gaussian Pyramid.



SIFT (Scale-Invariant Feature Transform)

- Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale



SIFT (Scale-Invariant Feature Transform)

- Keypoint Localization
- Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrastThreshold in OpenCV
- DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2x2 Hessian matrix (H) to compute the pricipal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function,
- If this ratio is greater than a threshold, called edgeThreshold in OpenCV, that keypoint is discarded. It is given as 10 in paper.
- So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points.

SIFT (Scale-Invariant Feature Transform)

- Orientation Assignment
- Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with \sigma equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching.

SIFT (Scale-Invariant Feature Transform)

//Keypoint Descriptor

- Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

//Keypoint Matching

- Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminaters around 90% of false matches while discards only 5% correct matches.

SURF
Speedup Robust Feature

- SIFT was comparatively slow and people needed more speeded-up version. In 2006, three people, Bay, H., Tuytelaars, T. and Van Gool, L, published another paper, “SURF: Speeded Up Robust Features” which introduced a new algorithm called SURF. As name suggests, it is a speeded-up version of SIFT.
- In short, SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

Object Detection in a Cluttered Scene Using SURF

This example presents an algorithm for detecting a specific object based on finding point correspondences between the reference and the target image. It can detect objects despite a scale change or in-plane rotation. It is also robust to small amount of out-of-plane rotation and occlusion.

This method of object detection works best for objects that exhibit non-repeating texture patterns, which give rise to unique feature matches. This technique is not likely to work well for uniformly-colored objects, or for objects containing repeating patterns. Note that this algorithm is designed for detecting a specific object, for example, the elephant in the reference image, rather than any elephant. For detecting objects of a particular category, such as people or faces, see vision.PeopleDetector and vision.CascadeObjectDetector.

SURF MATLAB

```
boxImage =
imread('stapleRem
over.jpg');
figure;
imshow(boxImage)
;
title('Image of a
Box');
```

SURF MATLAB

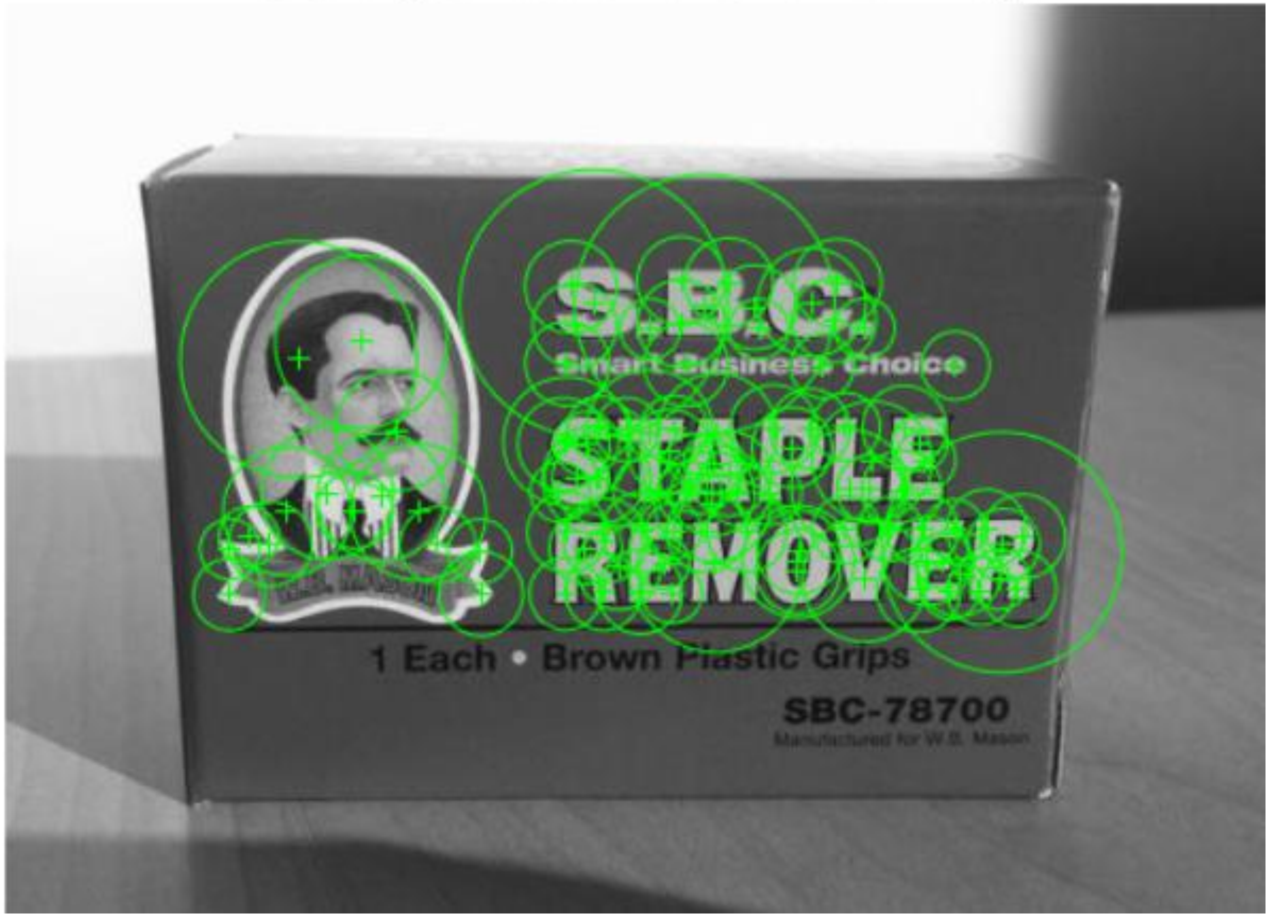
```
sceneImage =
imread('clutteredDesk.
jpg');
figure;
imshow(sceneImage);
title('Image of a
Cluttered Scene');
```

SURF MATLAB

```
%Detect feature points in both images.
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
%Visualize the strongest feature points found in the
%reference image.
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

SURF MATLAB

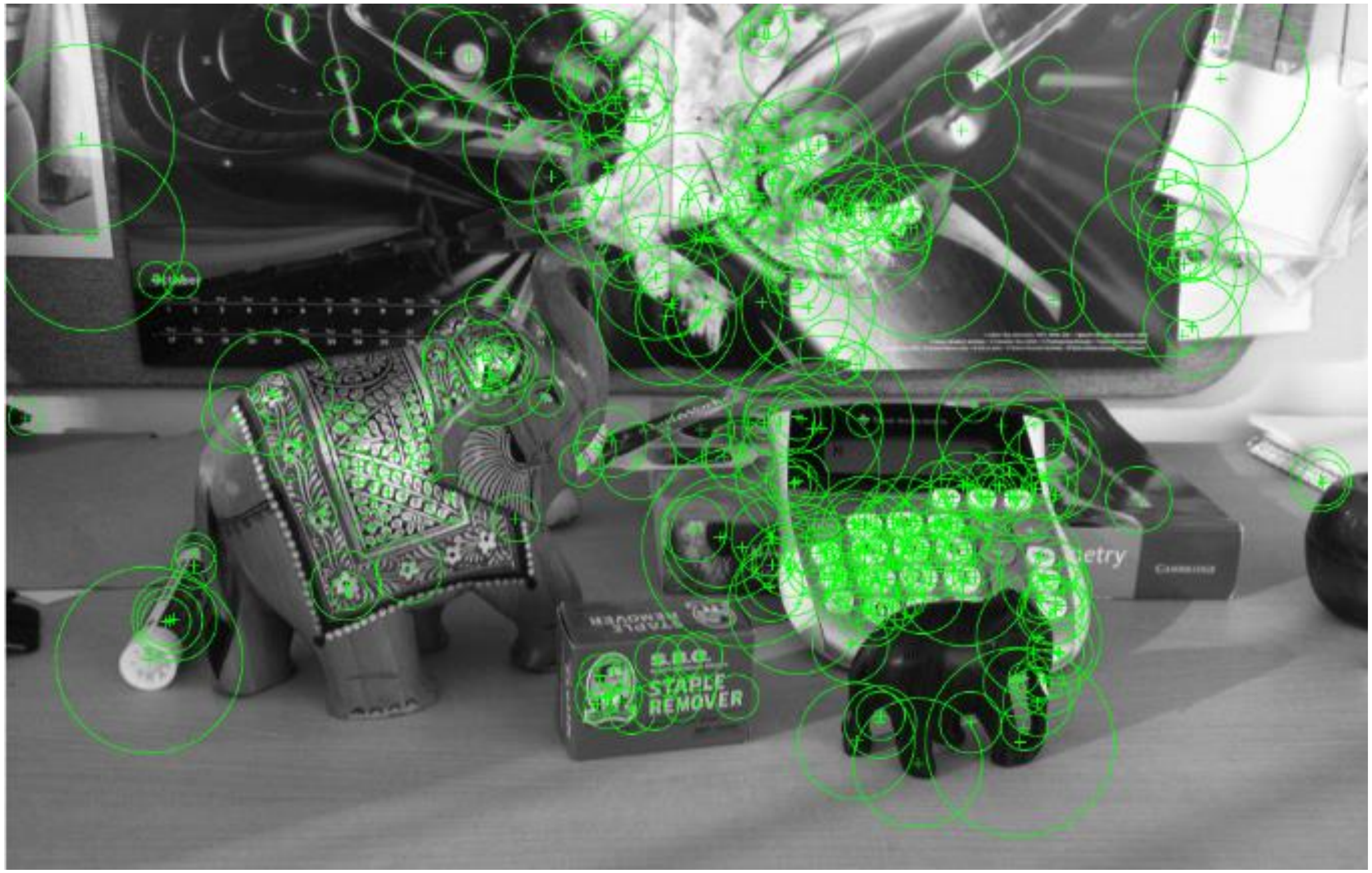
100 Strongest Feature Points from Box Image



SURF MATLAB

```
%Visualize the strongest feature points found in the target image.
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
plot(selectStrongest(scenePoints, 300));
```

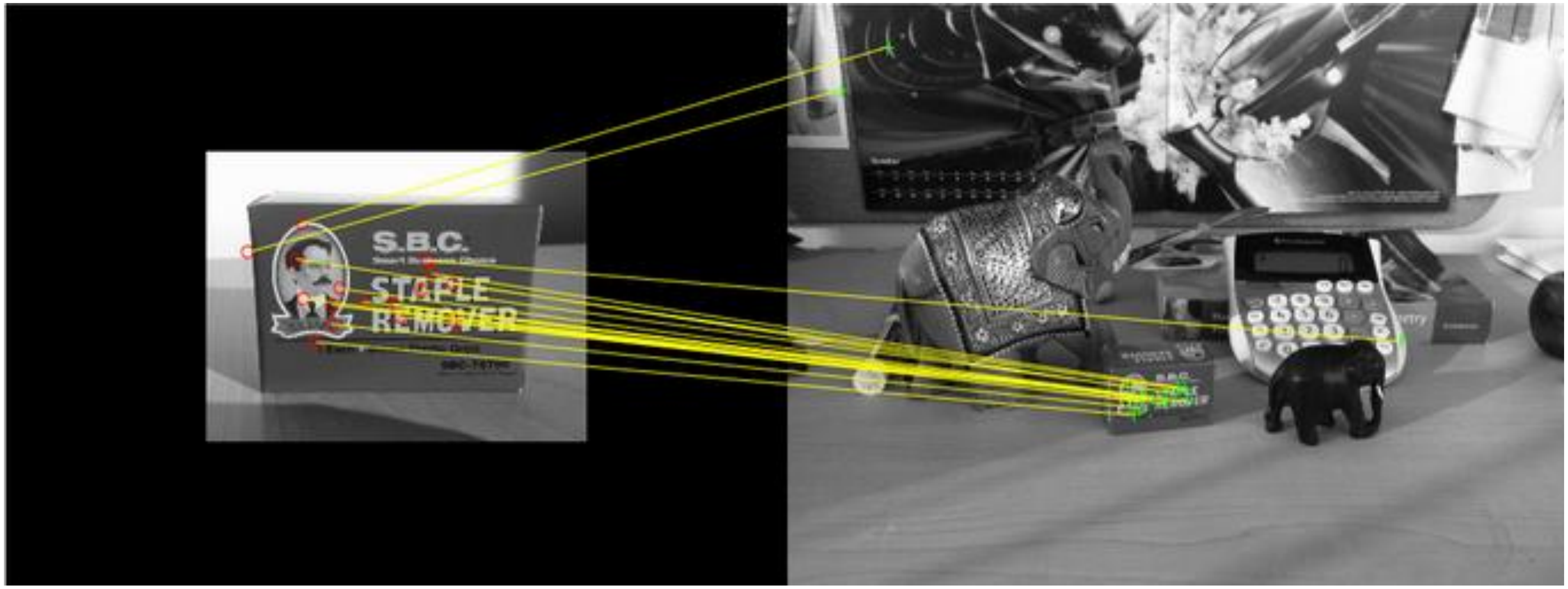
SURF MATLAB



SURF MATLAB

```
%Extract feature descriptors at the interest points in both images.
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage,
scenePoints);
%Find Putative Point Matches
%Match the features using their descriptors.
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
%Display putatively matched features.
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints,
...    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```

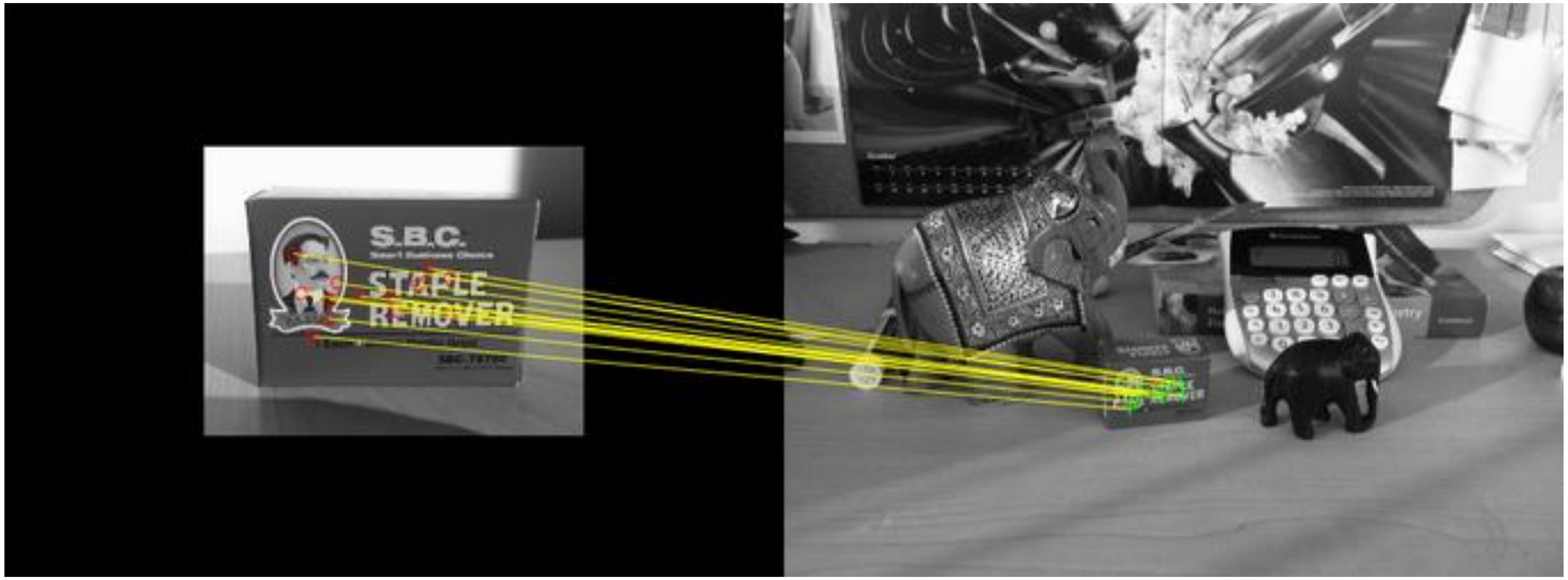
SURF MATLAB



SURF MATLAB

```
%Locate the Object in the Scene Using Putative Matches
%estimateGeometricTransform calculates the transformation
%relating the matched points, while eliminating outliers. This
%transformation allows us to localize the object in the scene.
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints,
matchedScenePoints, 'affine');
%Display the matching point pairs with the outliers removed
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```

SURF MATLAB

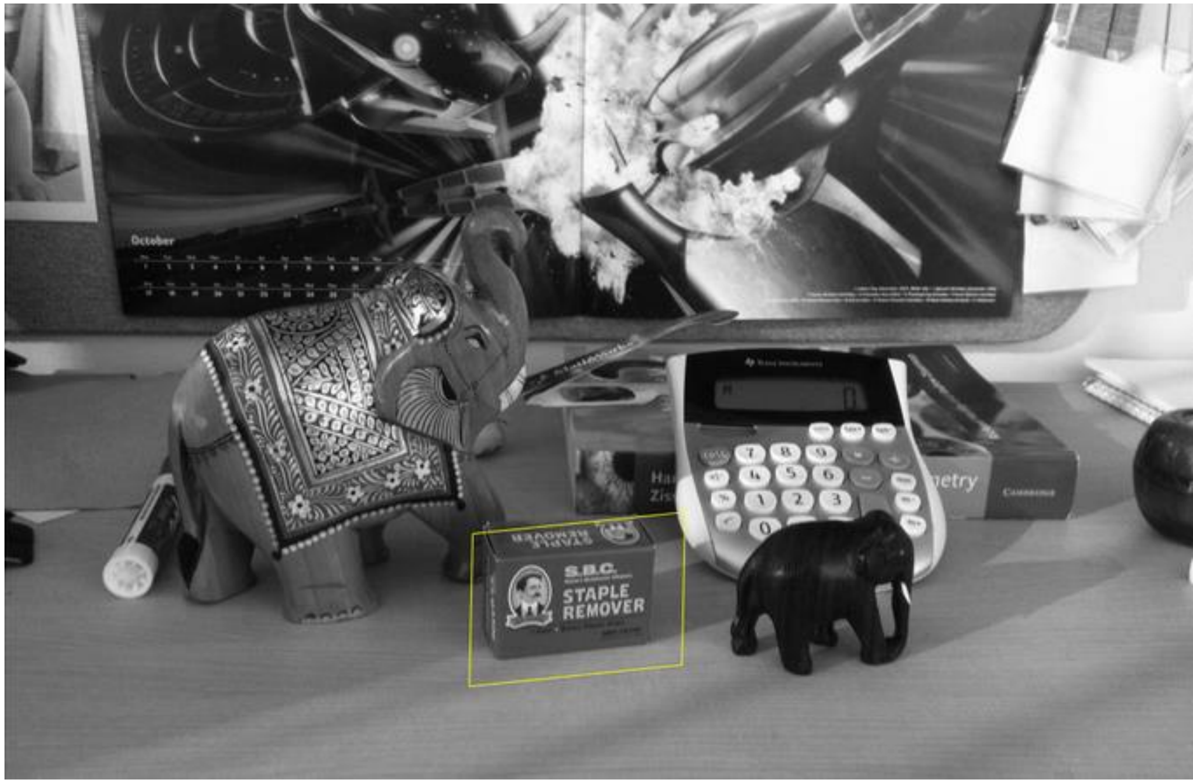


SURF MATLAB

```
%Get the bounding polygon of the reference image.
boxPolygon = [1, 1;...                              % top-left
        size(boxImage, 2), 1;...                    % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-
right
        1, size(boxImage, 1);...                    % bottom-left
        1, 1];                  % top-left again to close the polygon
%Transform the polygon into the coordinate system of
the target image. The transformed polygon indicates the
location of the object in the scene.
newBoxPolygon = transformPointsForward(tform,
boxPolygon);
```

SURF MATLAB

```
%Display the detected object.
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```



SURF MATLAB

```
%Detect a second object by using the same steps as before.
elephantImage = imread('elephant.jpg');
figure; imshow(elephantImage);
title('Image of an Elephant');
elephantPoints = detectSURFFeatures(elephantImage);
figure; imshow(elephantImage);
hold on;
plot(selectStrongest(elephantPoints, 100));
title('100 Strongest Feature Points from Elephant Image');
[elephantFeatures, elephantPoints] =
extractFeatures(elephantImage, elephantPoints);
elephantPairs = matchFeatures(elephantFeatures,
sceneFeatures, 'Ma
```

SURF MATLAB

```
matchedElephantPoints =
elephantPoints(elephantPairs(:, 1), :);
matchedScenePoints =
scenePoints(elephantPairs(:, 2), :);
figure; showMatchedFeatures(elephantImage,
sceneImage, matchedElephantPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including
Outliers)');
```

SURF MATLAB

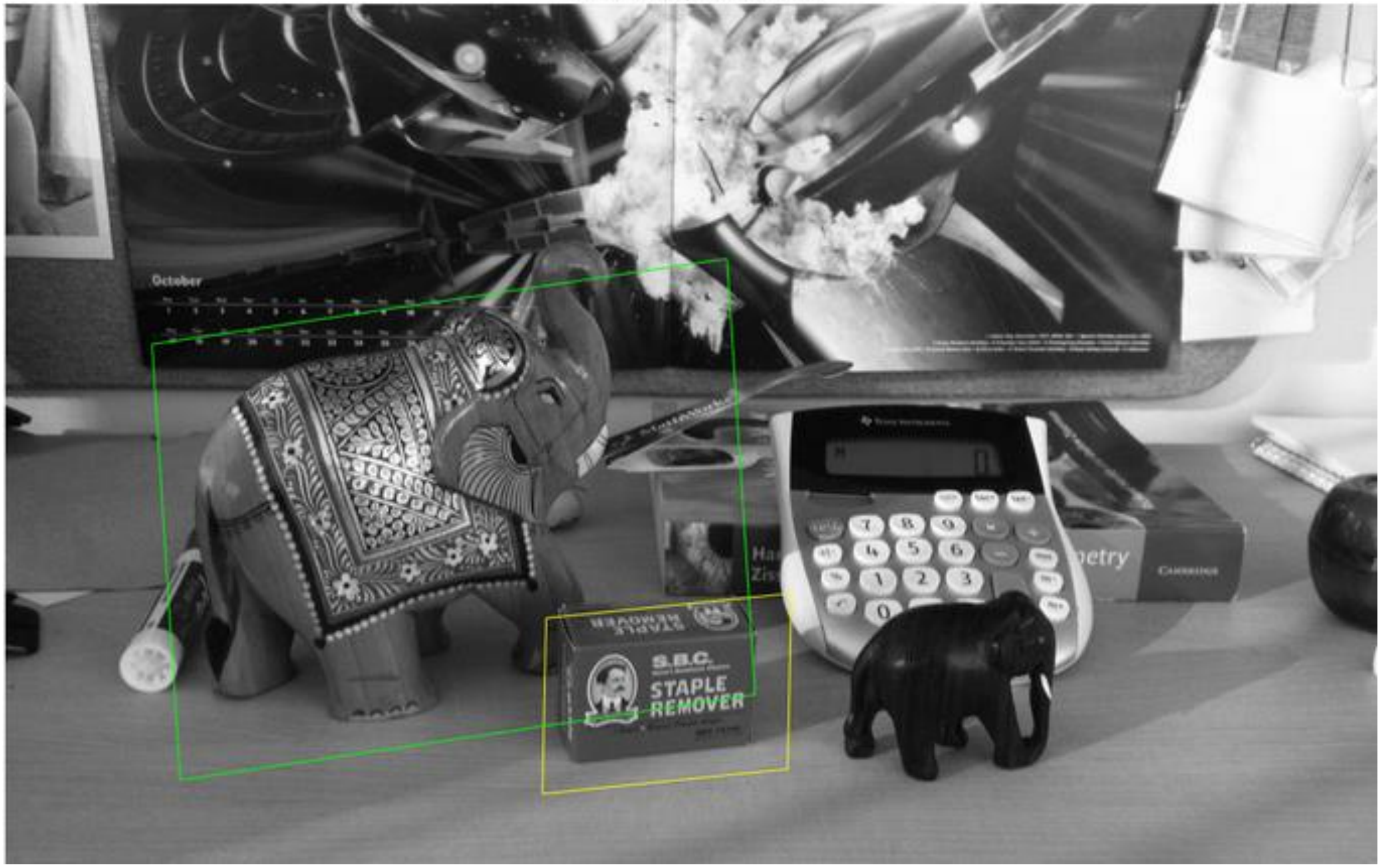
```
[tform, inlierElephantPoints, inlierScenePoints] = estimateGeometricTransform(matchedElephantPoints, matchedScenePoints, 'affine');

figure;

showMatchedFeatures(elephantImage, sceneImage, inlierElephantPoints, ...
    inlierScenePoints, 'montage');

title('Matched Points (Inliers Only)');
```

SURF MATLAB

```
elephantPolygon = [1, 1;...                                   % top-left
        size(elephantImage, 2), 1;...                          % top-right
        size(elephantImage, 2), size(elephantImage, 1);...  % bottom-right
        1, size(elephantImage, 1);...                          % bottom-left
        1,1];                    % top-left again to close the polygon
newElephantPolygon = transformPointsForward(tform,
elephantPolygon);
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
line(newElephantPolygon(:, 1), newElephantPolygon(:, 2), 'Color', 'g');
title('Detected Elephant and Box');
```

SURF MATLAB

Detected Elephant and Box



Find Homography SURF C++

```
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
using namespace cv;
int main( int argc, char** argv )
{
    Mat img_object = imread( argv[1],
CV_LOAD_IMAGE_GRAYSCALE );
  Mat img_scene = imread( argv[2],
CV_LOAD_IMAGE_GRAYSCALE );
```

Find Homography SURF C++

```
//Step 1: Detect the keypoints using SURF Detector
  int minHessian = 400;
  SurfFeatureDetector detector( minHessian );
  std::vector<KeyPoint> keypoints_object,
keypoints_scene;
  detector.detect( img_object, keypoints_object );
  detector.detect( img_scene, keypoints_scene );
  //Step 2: Calculate descriptors (feature vectors)
  SurfDescriptorExtractor extractor;
  Mat descriptors_object, descriptors_scene;
  extractor.compute( img_object, keypoints_object,
descriptors_object );
  extractor.compute( img_scene, keypoints_scene,
descriptors_scene );
```

Find Homography SURF C++

```
//-- Step 3: Matching descriptor vectors using FLANN matcher
  FlannBasedMatcher matcher;
  std::vector< DMatch > matches;
  matcher.match( descriptors_object, descriptors_scene,
matches );
  double max_dist = 0; double min_dist = 100;
  //-- Quick calculation of max and min distances between
keypoints
  for( int i = 0; i < descriptors_object.rows; i++ )
  { double dist = matches[i].distance;
    if( dist < min_dist ) min_dist = dist;
    if( dist > max_dist ) max_dist = dist;
  }
  printf("-- Max dist : %f \n", max_dist );
  printf("-- Min dist : %f \n", min_dist );
```

Find Homography SURF C++

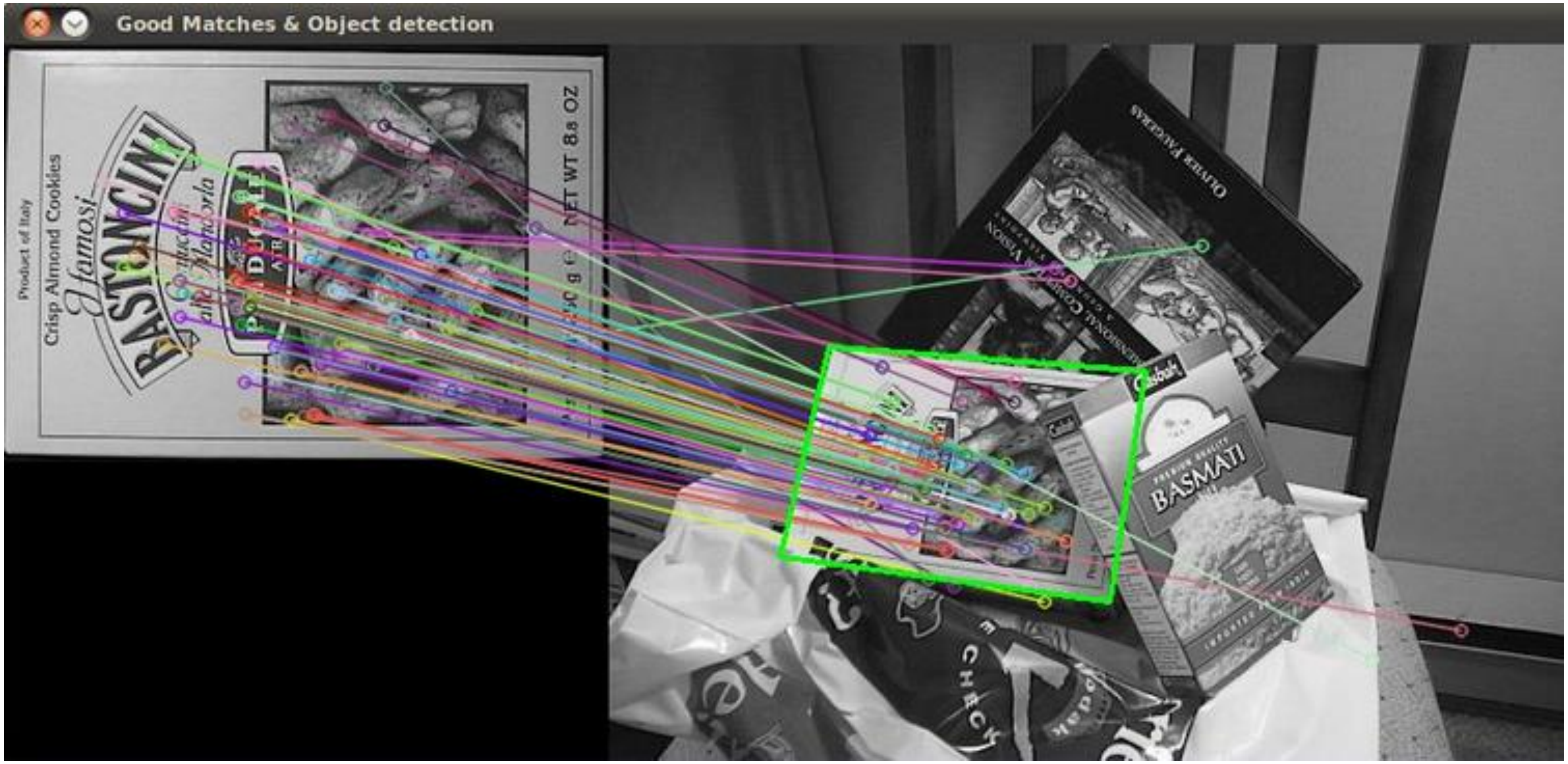
```
//-- Draw only "good" matches (i.e. whose distance is less
than 3*min_dist )
  std::vector< DMatch > good_matches;
  for( int i = 0; i < descriptors_object.rows; i++ )
  { if( matches[i].distance < 3*min_dist )
     { good_matches.push_back( matches[i]); }
  }
  Mat img_matches;
  drawMatches( img_object, keypoints_object, img_scene,
keypoints_scene, good_matches, img_matches,
Scalar::all(-1), Scalar::all(-1), vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
  //-- Localize the object
  std::vector<Point2f> obj;
  std::vector<Point2f> scene;
```

Find Homography SURF C++

```
for( int i = 0; i < good_matches.size(); i++ )
  {
    //-- Get the keypoints from the good matches
    obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );
    scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt
);
  }
  Mat H = findHomography( obj, scene, CV_RANSAC );
  //-- Get the corners from the image_1 ( the object to be "detected" )
  std::vector<Point2f> obj_corners(4);
  obj_corners[0] = cvPoint(0,0); obj_corners[1] = cvPoint(
img_object.cols, 0 );
  obj_corners[2] = cvPoint( img_object.cols, img_object.rows );
obj_corners[3] = cvPoint( 0, img_object.rows );
  std::vector<Point2f> scene_corners(4);
```

```
perspectiveTransform( obj_corners, scene_corners, H);
  //-- Draw lines between the corners (the mapped object in the scene - image_2 )
  line( img_matches, scene_corners[0] + Point2f( img_object.cols, 0), scene_corners[1] + Point2f( img_object.cols, 0), Scalar(0, 255, 0), 4 );
  line( img_matches, scene_corners[1] + Point2f( img_object.cols, 0), scene_corners[2] + Point2f( img_object.cols, 0), Scalar( 0, 255, 0), 4 );
  line( img_matches, scene_corners[2] + Point2f( img_object.cols, 0), scene_corners[3] + Point2f( img_object.cols, 0), Scalar( 0, 255, 0), 4 );
  line( img_matches, scene_corners[3] + Point2f( img_object.cols, 0), scene_corners[0] + Point2f( img_object.cols, 0), Scalar( 0, 255, 0), 4 );
  //-- Show detected matches
  imshow( "Good Matches & Object detection", img_matches );
  waitKey(0);
  return 0;
  }
```

Find Homography SURF C++


Good Matches & Object detection
Product of Italy
Crisp Almond Cookies
Famosi
BASTONCINI
NET WT 8.8 OZ
BASMATI
Olivier Faugeras