Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 13

Image Features

SIFT descriptor, feature descriptor matching

Objects and Interest Points (IPs)

1. Feature detection
   Extract interest points
   (unique image regions)

2. Feature description
   Calculate local (invariant) descriptors

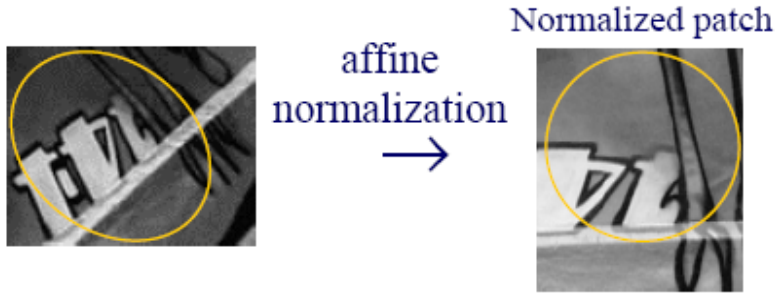
3. Feature matching / feature tracking
   Find correspondences

4. Find similar image regions/objects

Credit: Markus Vincze, Technische Universität Wien

Local Descriptors

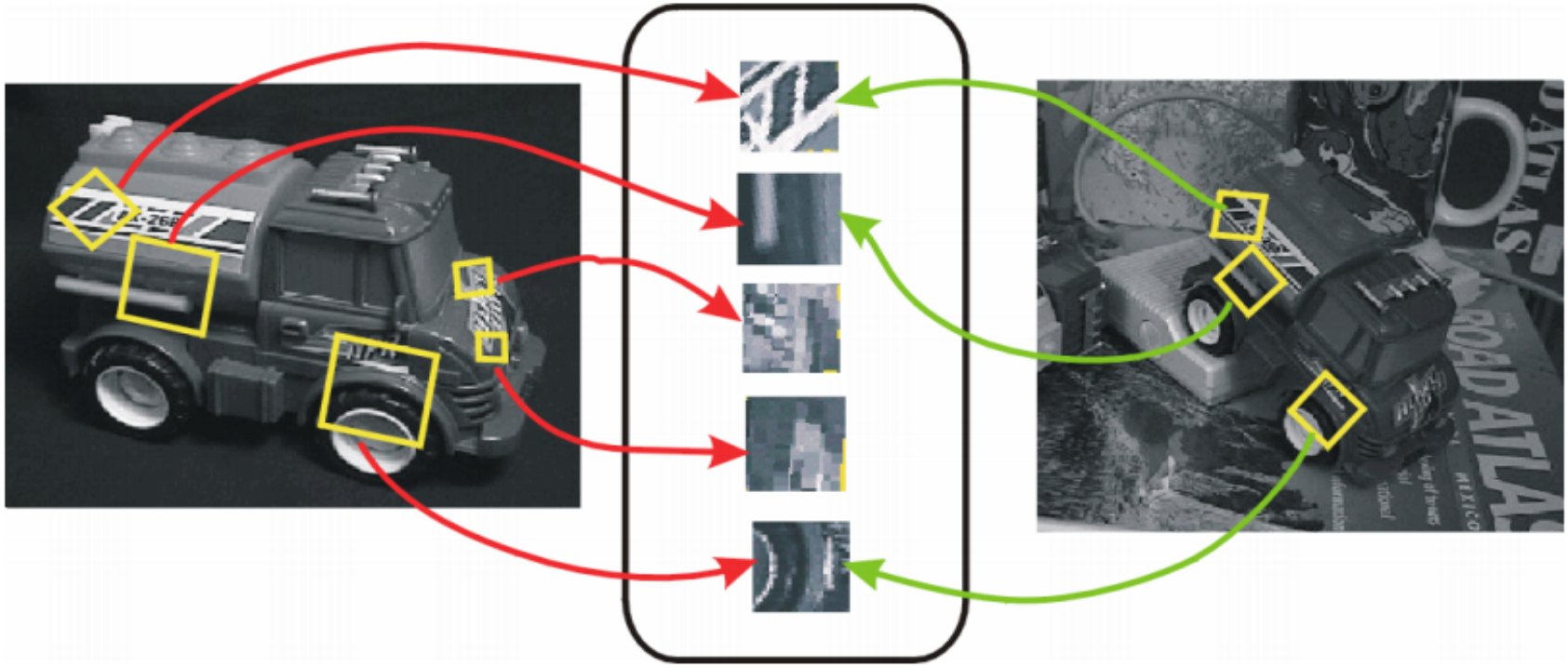
- Distinctive
- Invariant to geometric and photometric transforms
- Robust to viewing angle, illumination, …
- Example descriptors
  - Sampled image patch (template)
  - Gradient orientation histogram – SIFT (Lowe)
  - Shape context [Belongie et al. '02]
  - PCA-SIFT [Ke and Sukthankar '04]
  - HOG (Histogram of Oriented Gradients) [Dalal CVPR'05]
  - SURF (fast approximate SIFT) [Bay ECCV'06, Cornelis CVGPU'08]



Credit: Markus Vincze, Technische Universität Wien

Invariant Local Features

Image region are transformed into feature coordinates such that they are invariant to translation, rotation, size and other imaging parameters (partly invariant to illumination)



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Detection and description of local features with reasonable invariance to

- Rotation
- Scale change
- Change in viewing angle
- Change in illumination
- Noise

David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* 60, 2 (2004), pp. 91-110.

Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Overview

1. Scale Space Extrema Detection
2. Accurate Keypoint Location
3. Keypoint Orientation Assignment
4. Keypoint Descriptors – The Local Image Descriptor

Scale-Invariant Feature Transform - SIFT

Overview

1. Scale Space Extrema Detection

   - Search over all scales and locations in scale space
   - Using a Difference-of-Gaussian (DoG) function to identify potential interest points that are invariant to scale and orientation
   - Locate Extrema in DoG correspond to potential interest points

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

Overview

2. Accurate Keypoint Location

   - Sub-pixel accuracy
   - Remove low contrast features
   - Remove features primarily along an edge

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

Overview

3. Keypoint Orientation Assignment

- One (or more) orientations are assigned to each keypoint location
  - based on local image gradient directions

- All subsequent operations are performed on image data that has been transformed relative to the to these assigned
  - orientation,
  - scale,
  - location

Scale-Invariant Feature Transform - SIFT

Overview

4. Keypoint Descriptors – The Local Image Descriptor

   - Measure local image gradients at the selected scale in the region around each keypoint

   - Transform into a representation that allows for significant levels of local shape distortion and change in illumination

Scale-Invariant Feature Transform - SIFT

Overview

- Matching Descriptors
  - neglecting poor matches
- Applications

Scale-Invariant Feature Transform - SIFT

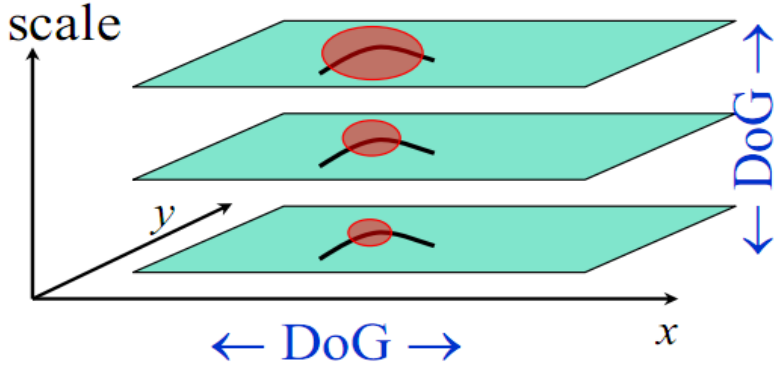
Overview

- SIFT generates large numbers of features
- Densely cover the image over the full range of scales and locations
- A typical image of size 500x500 pixels will give rise to about 2000 stable features

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

- SIFT (Lowe)[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale



[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

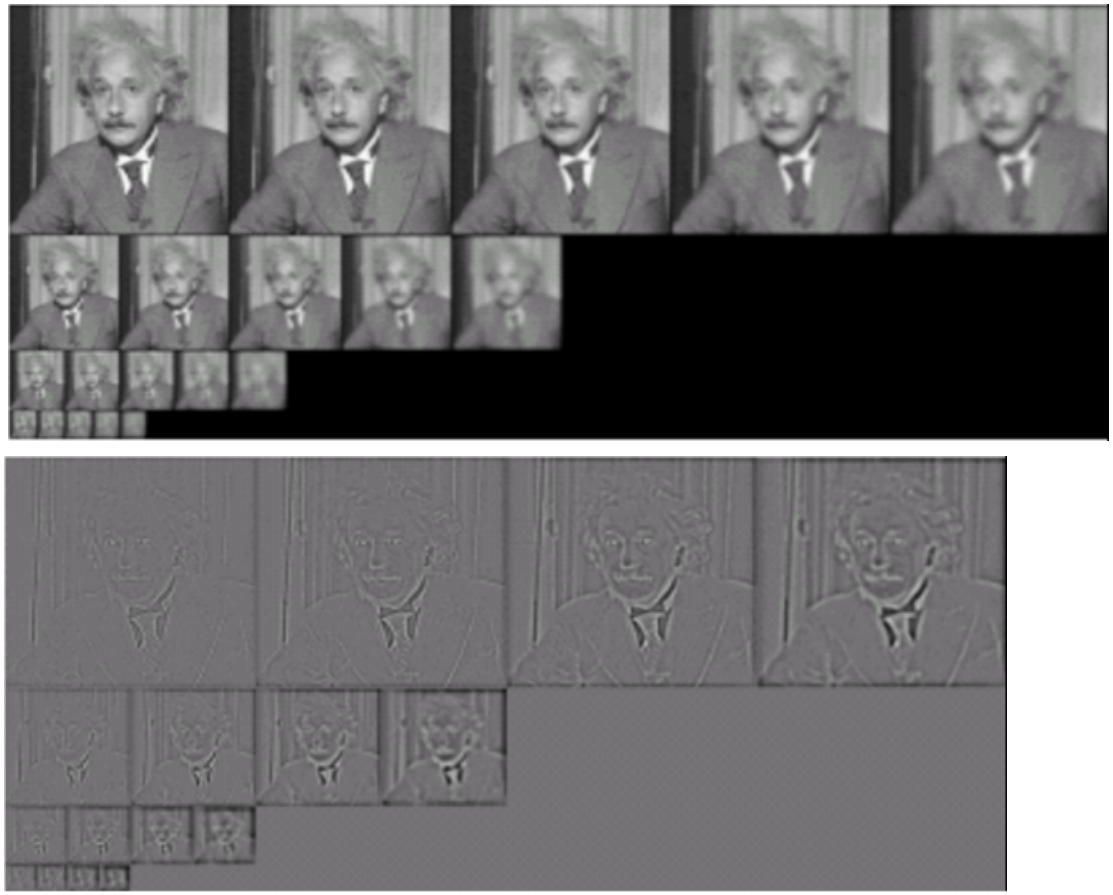
Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

Difference of Gaussian (DoG)

- Gaussian smoothed image at size (scale) σ and $s\sigma$
- Grouped by octaves (i.e. doubling of σ)
- s sets the number of images per octave
- DoG images grouped by octave



Credit: Markus Vincze, Technische Universität Wien

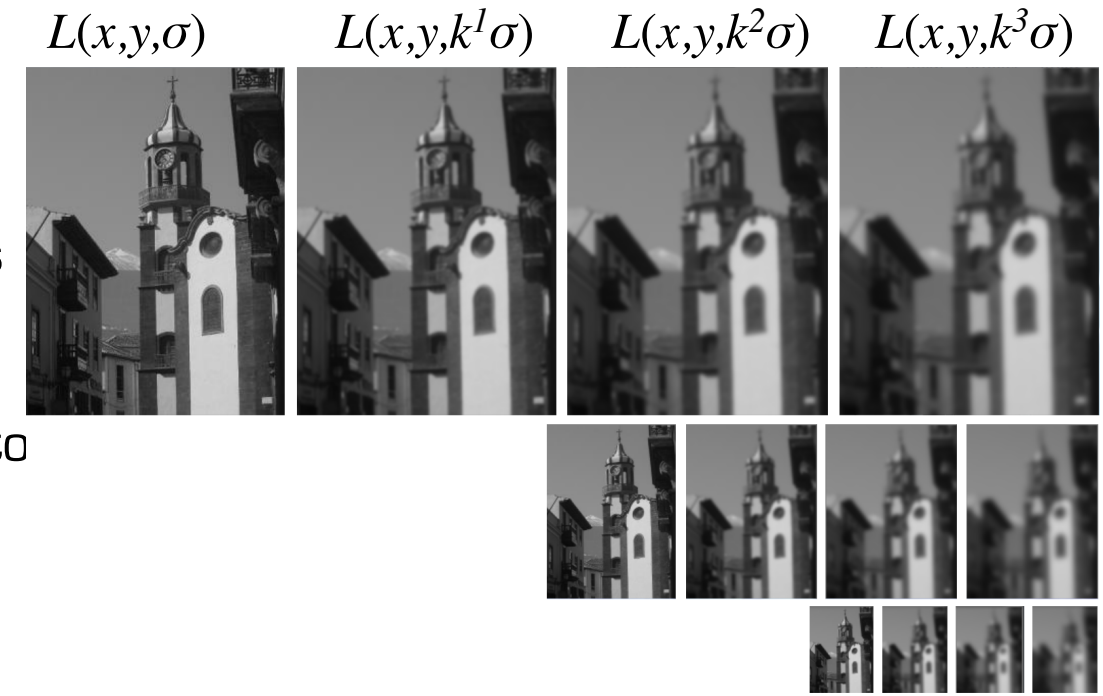
Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

Step 1: Form a scale space representation of the image

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

- Applied in different octaves of scale space
- Each octave corresponds to a doubling of σ
- $k = 2^{1/s}$ where s is the number of intervals in an octave



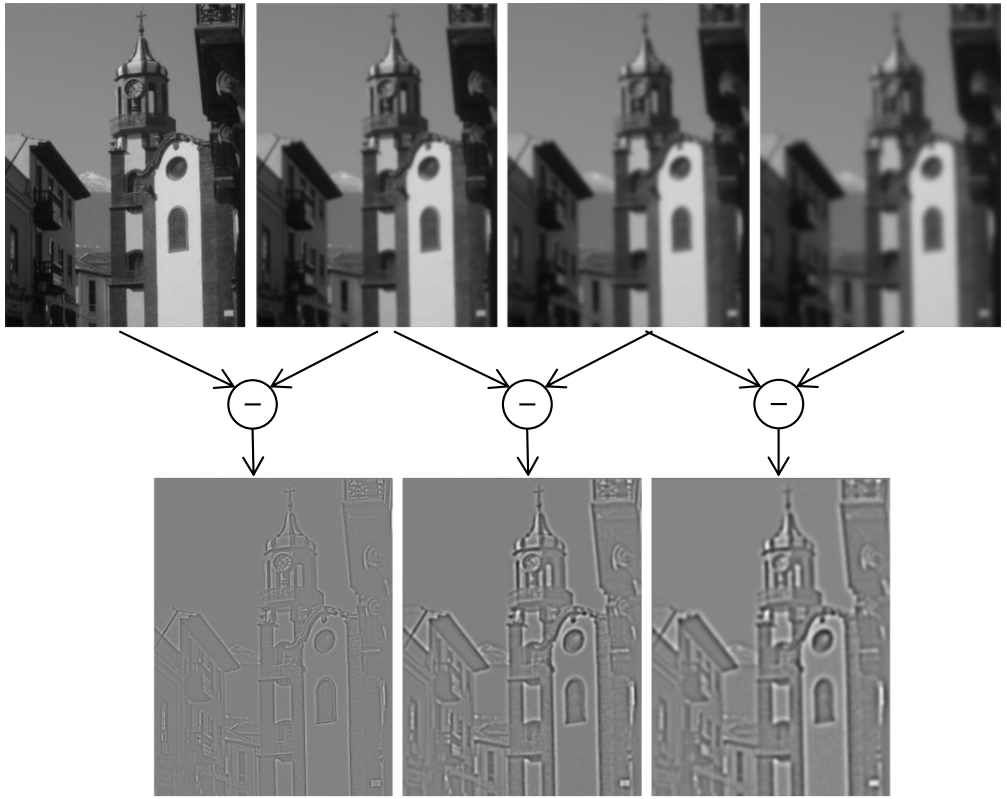
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

Step 2: Form Difference-of-Gaussian (DoG) images across scale space

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



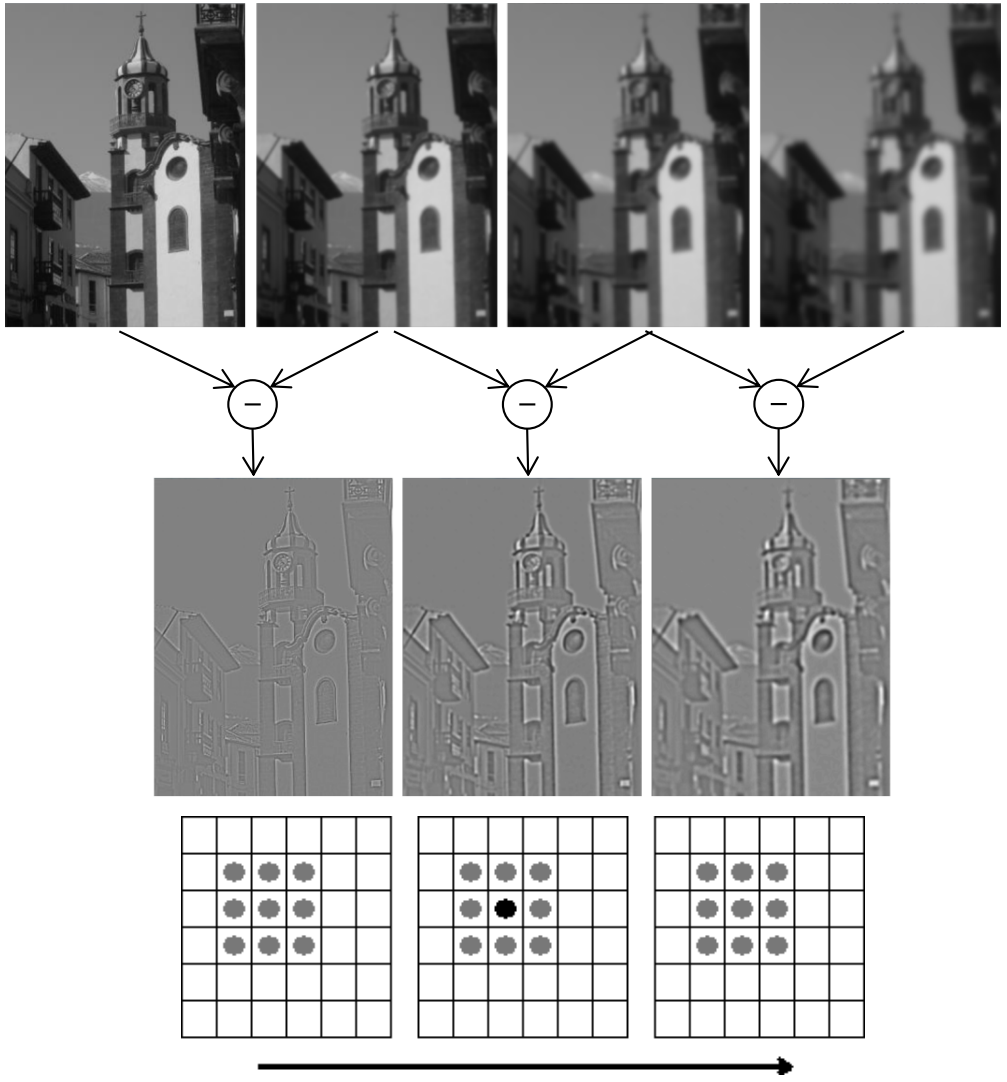
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

Step 3: Find extrema in DoG

- Extremum is centre point that is min or max of
- local 3x3 region in current DoG
- and in adjacent scales
- in any octave
- hence scale invariance



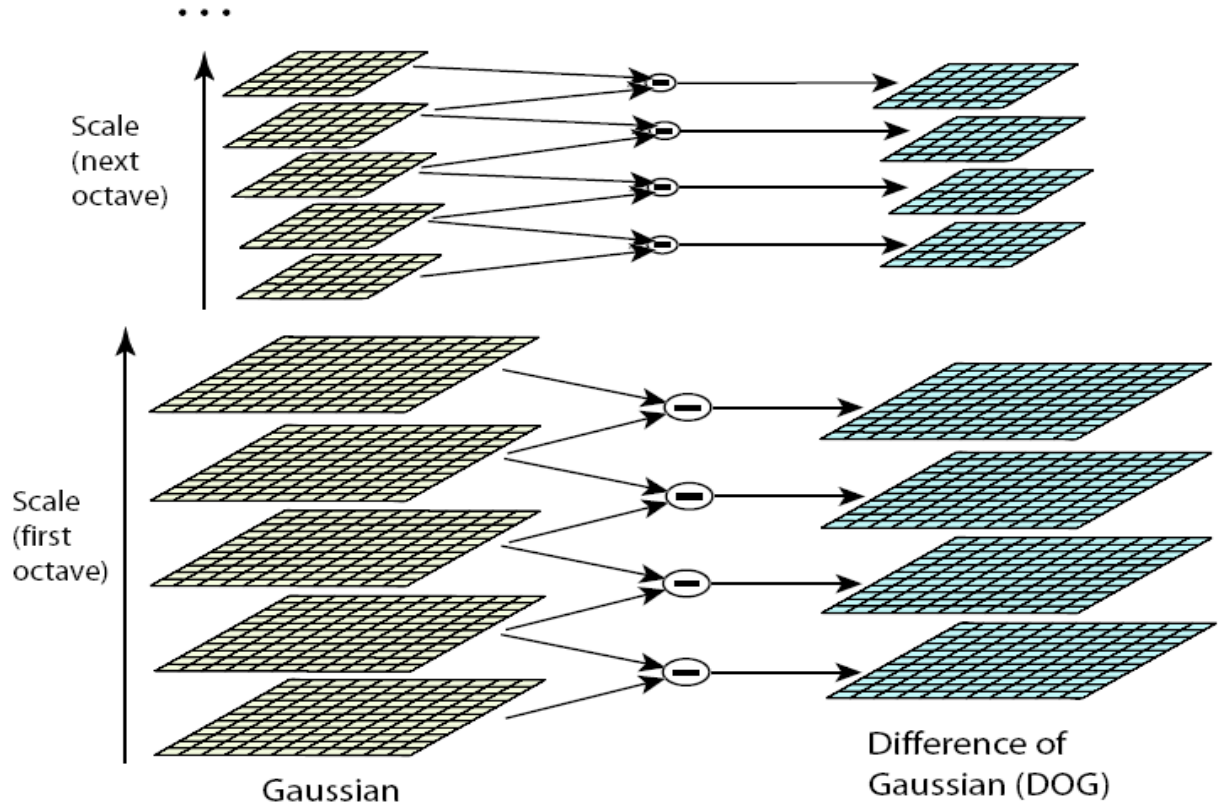
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

   Difference of Gaussian (DoG)

   - Search maximum in DoG image pyramid
   - For every octave s sub-scales (= gradually smoothed images within octave)



Credit: Markus Vincze, Technische Universität Wien

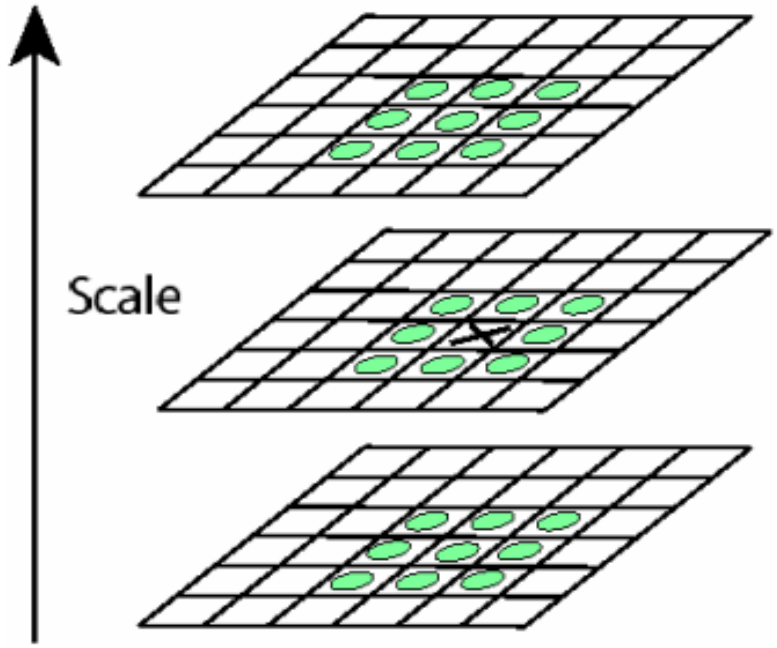
Scale-Invariant Feature Transform - SIFT

1. Scale Space Extrema Detection

   Keypoints: extrema in Difference of Gaussian (DoG)

   – Every pixel in the DoG images is compared with its 8 neighbours and the 9 neighbours of the neighbouring scale Gauss images (scale space images) ... 26 in total

   – Interest points (key points, frames) are local maxima or minima of the DoG over all sizes



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location

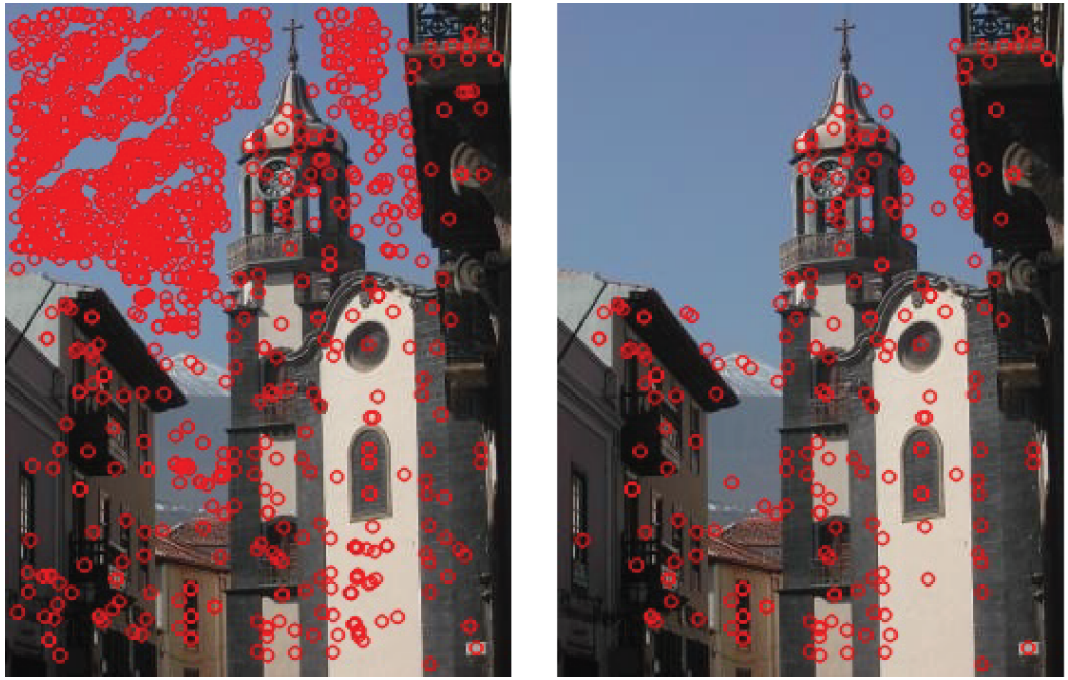
- Initial location and scale taken from central point
- Locate keypoints more precisely
  - Model data locally using a 3D quadratic
  - Locate interpolated maximum/minimum

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location

- Discard low contrast keypoints
- Evaluated from the curvature of the 3D quadratic



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location

- Discard poorly localised keypoints
- e.g. along an edge where curvature and ”cornerness” is low
- Evaluated from the ratio of the eigenvalues, as follows

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location

- Discard poorly localised keypoints … these have a large principal curvature across the edge but a small one in the perpendicular direction (low cornerness)
- Evaluate this from the ratio of the eigenvalues (α, β) so that it is not necessary to compute the eigenvalues explicitly …
- Consider the Hessian $\mathbf{H}$

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

Let α and β be the largest and smallest eigenvalues, respectively

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location

- Discard poorly localised keypoints … these have a large principal curvature across the edge but a small one in the perpendicular direction (low cornerness)
- Evaluate this from the ratio of the eigenvalues (α, β)
  so that it is not necessary to compute the eigenvalues explicitly …

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

Minimum when $\alpha = \beta$ and increases with r

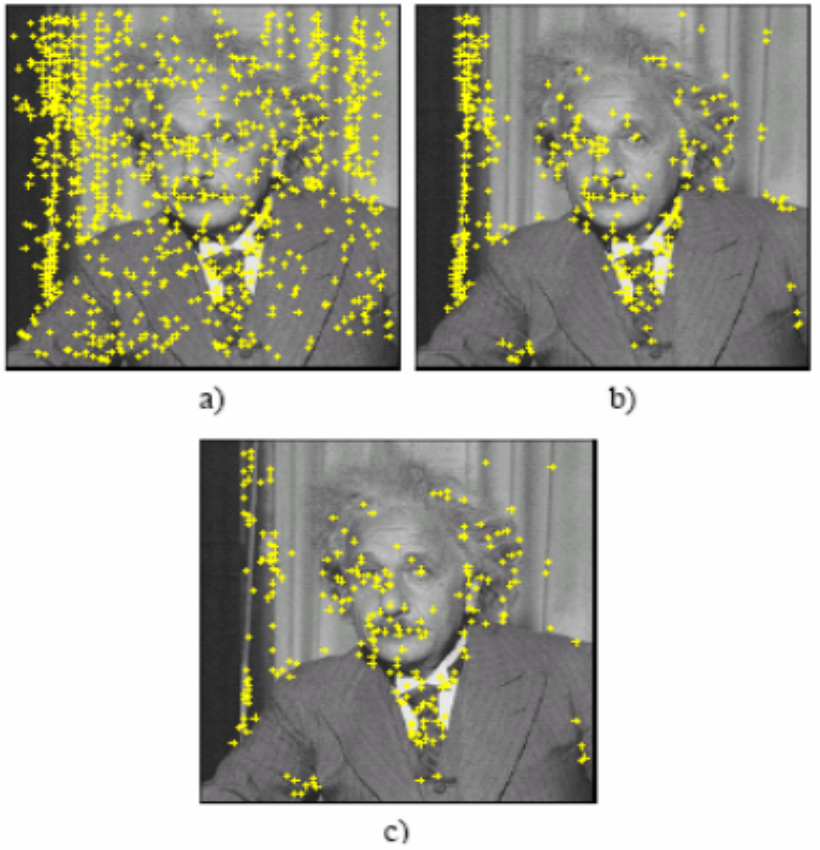
Let $r = \alpha / \beta$ the ratio between the largest and smallest eigenvalues: $\alpha = r\,\beta$

Accept keypoints for which $\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$

r = 10 in Lowe's paper

Scale-Invariant Feature Transform - SIFT

2. Accurate Keypoint Location - Summary
   - From all interest points (a)
   - Points with low contrast are eliminated (b)
   - Points along edges are eliminated (c)
   - Points are interpolated to obtain the position more accurately



Scale-Invariant Feature Transform - SIFT

3. Keypoint Orientation Assignment

– For orientation invariance we describe the keypoint w.r.t. its principal orientation

  • there might be more than one

– Compute the gradient magnitude $m(x, y)$ and orientation $\theta(\mathrm{x}, y)$ at each point in a region around the keypoint (e.g. a16x16 pixel region)

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

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

3. Keypoint Orientation Assignment

Note: Lowe uses 16x16 pixel region rather than the 8x8 pixel region shown

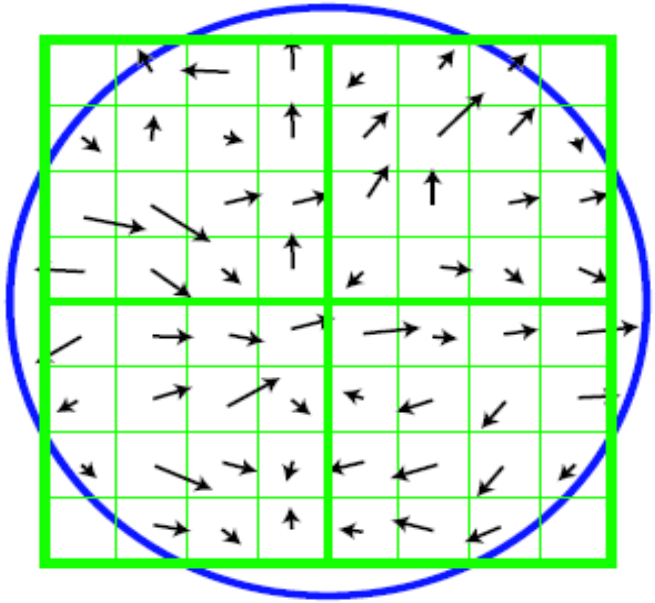


Image gradients

Scale-Invariant Feature Transform - SIFT

3. Keypoint Orientation Assignment

   – Create an orientation histogram (36 bins … 1 bin per 10°)

      • Weight each sample by gradient magnitude

      • Weight each sample by a Gaussian window function with σ = 1.5 the σ of the scale of the keypoint

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Scale-Invariant Feature Transform - SIFT

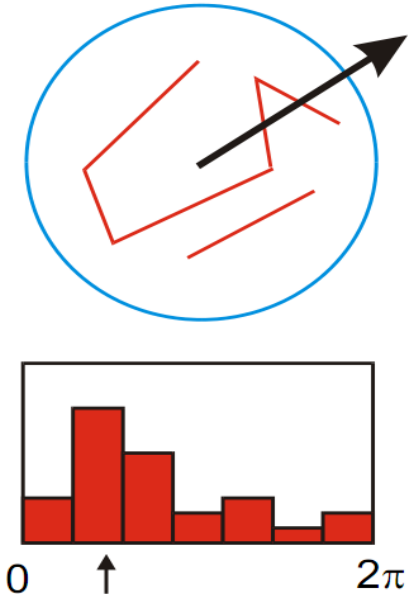
3. Keypoint Orientation Assignment

   - Peaks in the orientation histogram correspond to dominant directions of local gradients
   - The highest peak in the histogram is detected
   - Any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation
   - Thus, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations
   - Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy
     - Resultant pitted peak position gives orientation with accuracy greater than 10 degrees

Scale-Invariant Feature Transform - SIFT

3. Keypoint Orientation Assignment - Summary

- Determine a reference orientation
- Calculate the orientation histogram in region around the interest point
- Maxima in histogram define dominant gradient direction → reference orientation
  - If there is more than one maximum, store more than one keypoint
- All other properties of a key point are relative to this orientation
  - This yields invariance to orientation

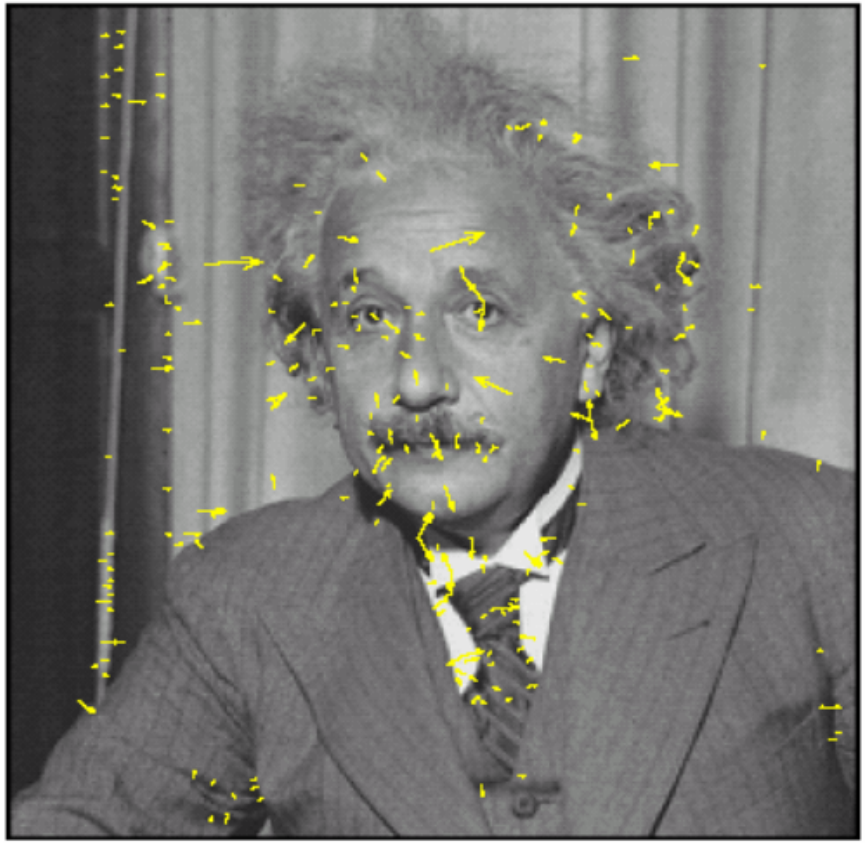


Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

3. Keypoint Orientation Assignment - Summary

- Orientation
- Size (length of arrows)



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

4. Keypoint Descriptors – The Local Image Descriptor

   – Compute a descriptor for the local image region

   – that is highly distinctive

   – yet is as invariant as possible to remaining variations

      • such as change in illumination or 3D viewpoint.

Scale-Invariant Feature Transform - SIFT

4. Keypoint Descriptors – The Local Image Descriptor

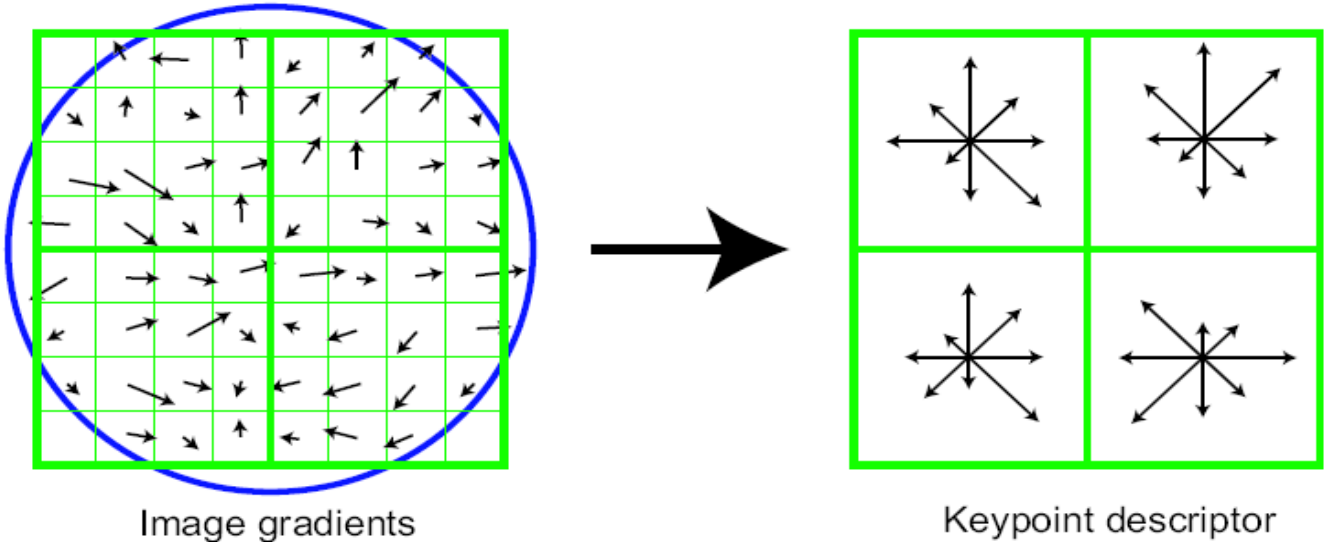
   - Use Gaussian image at closest scale to the keypoint
   - Sample points around the keypoint
   - Compute gradients and orientations
   - Rotate gradient orientations by keypoint orientation
   - Divide region into subregions
   - Create histograms (8 bins) for subregions
   - Weight samples by gradient
   - Weight samples by location
     - Gaussian σ equal to one half the width of the descriptor window
   - Distribute value of each gradient sample into adjacent histogram bins
     - using trilinear interpolation

Scale-Invariant Feature Transform - SIFT

4. Keypoint Descriptors – The Local Image Descriptor

   – Compute several orientation histograms

   – Note: Lowe uses 16x16 pixel region rather than the 8x8 pixel region shown and 4x4 keypoint descriptor rather than the 2x2 descriptor shown



Credit: Markus Vincze, Technische Universität Wien

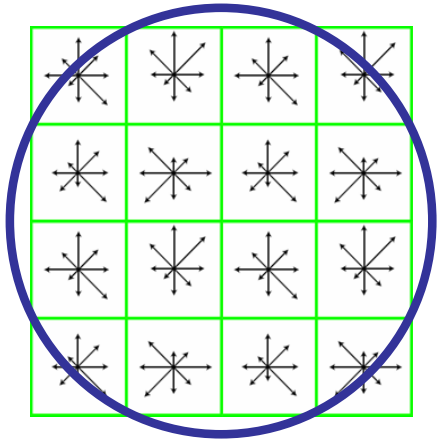
Scale-Invariant Feature Transform - SIFT

4. Keypoint Descriptors – The Local Image Descriptor
   SIFT Descriptor

   – Vector with 128-D description of the region around the key point

      • 8 orientations x 4 x 4 histograms = 128 dimensions
      • Entries are the arrows in the figure below

   – Vector is normalised to compensate for intensity changes in images



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Advantages

- **Local**: features are local and robust to occlusion and complex scenes with clutter
  - No need to segment object from background
  - Depends on finding characteristic points
- **Robust**: histograms allow for shifts of a few pixel
- **Distinctive**: individual feature vectors stored in large data bases and used in fast search methods (128 dimensions!)
- **Quantity**: many features also for small textured objects
- **Efficient**: fast calculation (< second, frame rate with GPU-SIFT)

Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Recognition of Planar Objects

- Planar surfaces recognised robustly with up to 60° rotation away from camera
- Affine transformation estimates the perspective projection
- 3 points sufficient to obtain full object pose

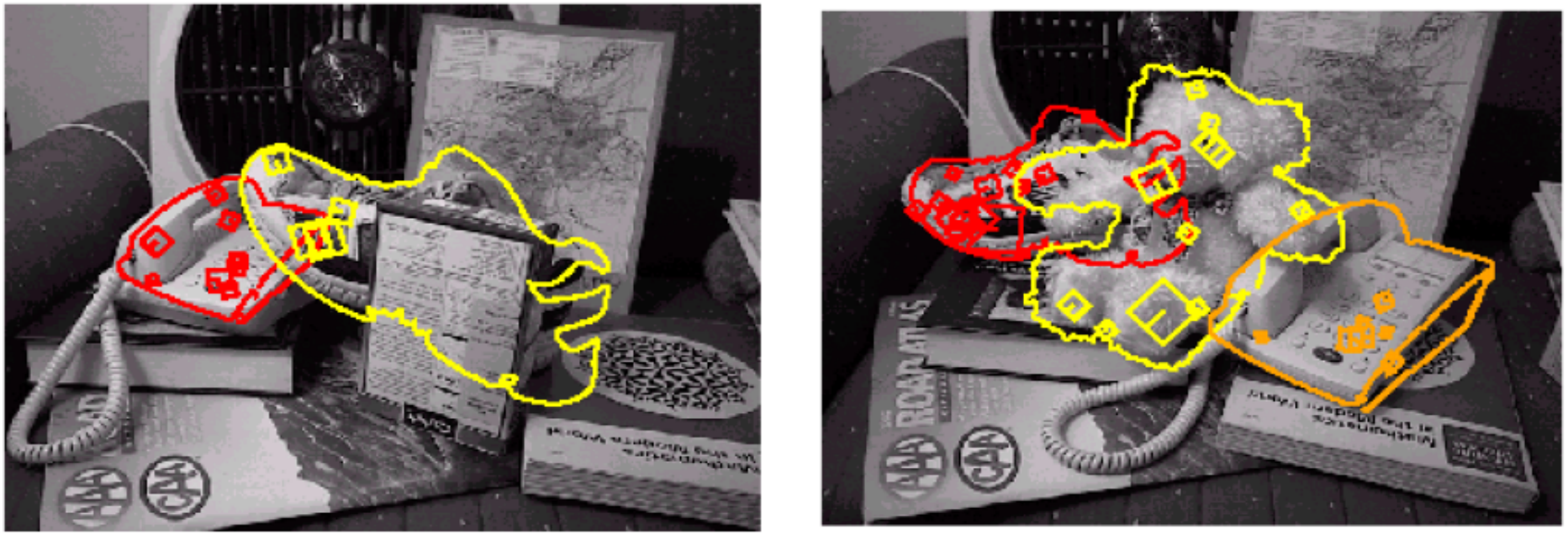
  → good if partially occluded

[Lowe 2004]

Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Recognition with occlusion



[Lowe 2004]

Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Recognition and segmentation

- Initialisation of object surface with dense features
- Iterative search for features with affine refinement

[Ferrari 04]

Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

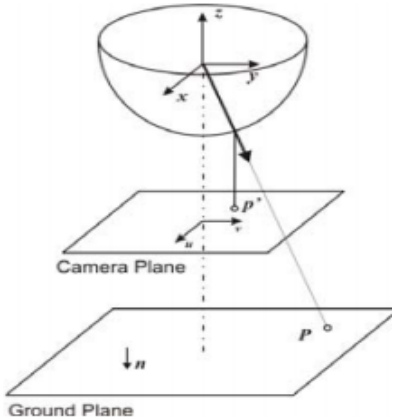
Recognition of places

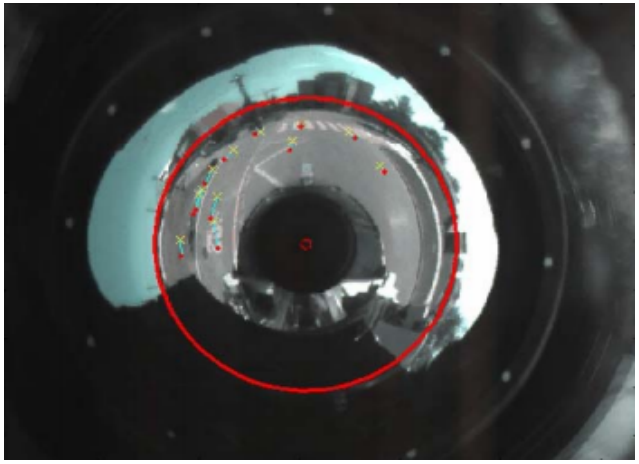
Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Estimating egomotion of a vehicle

- Calibrated omni-directional camera
- SIFT and estimation of camera motion [Triggs'98]
- [Scaramuzza and Siegwart, ICVS'08]



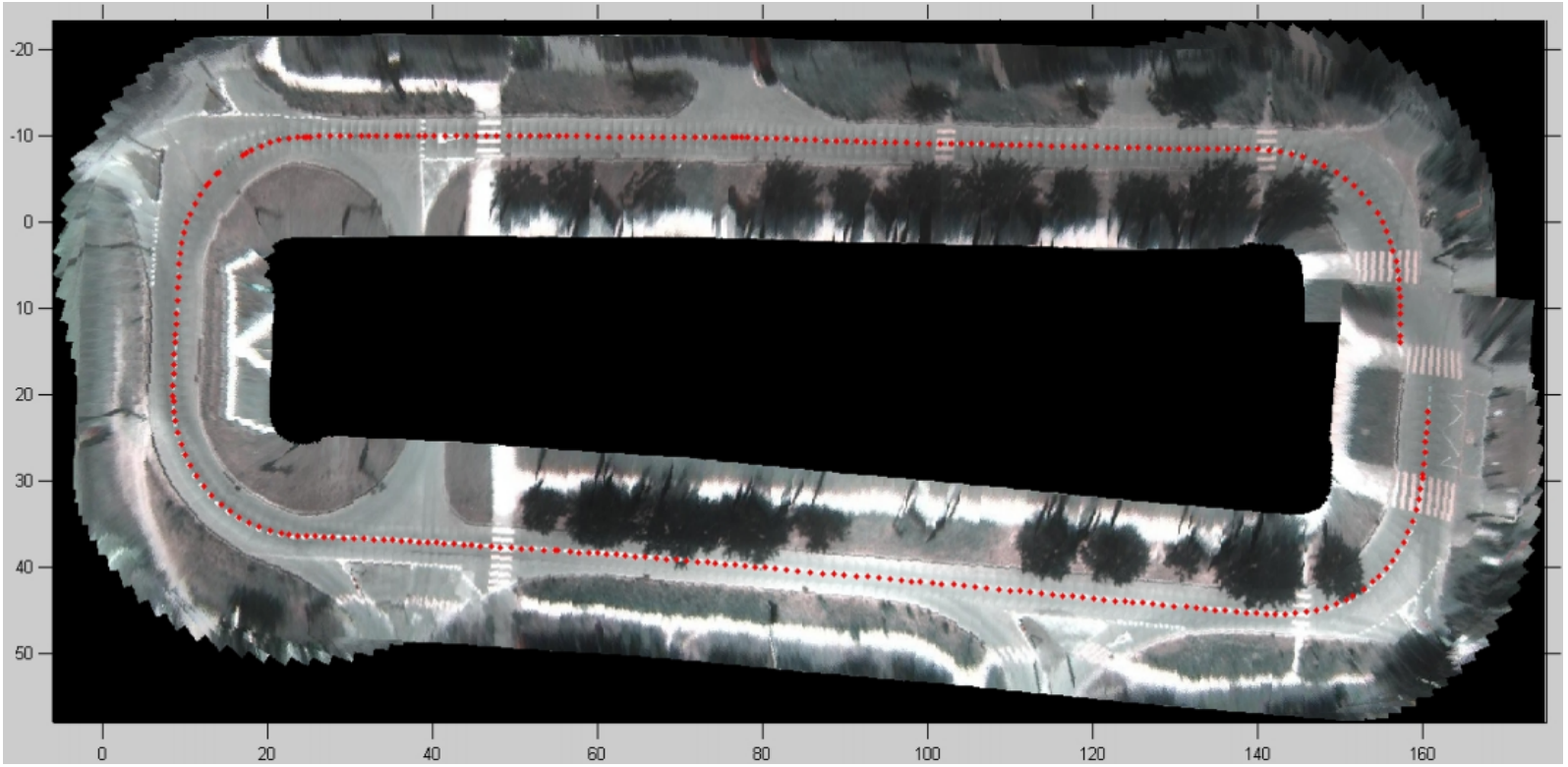


Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Estimating egomotion of a vehicle

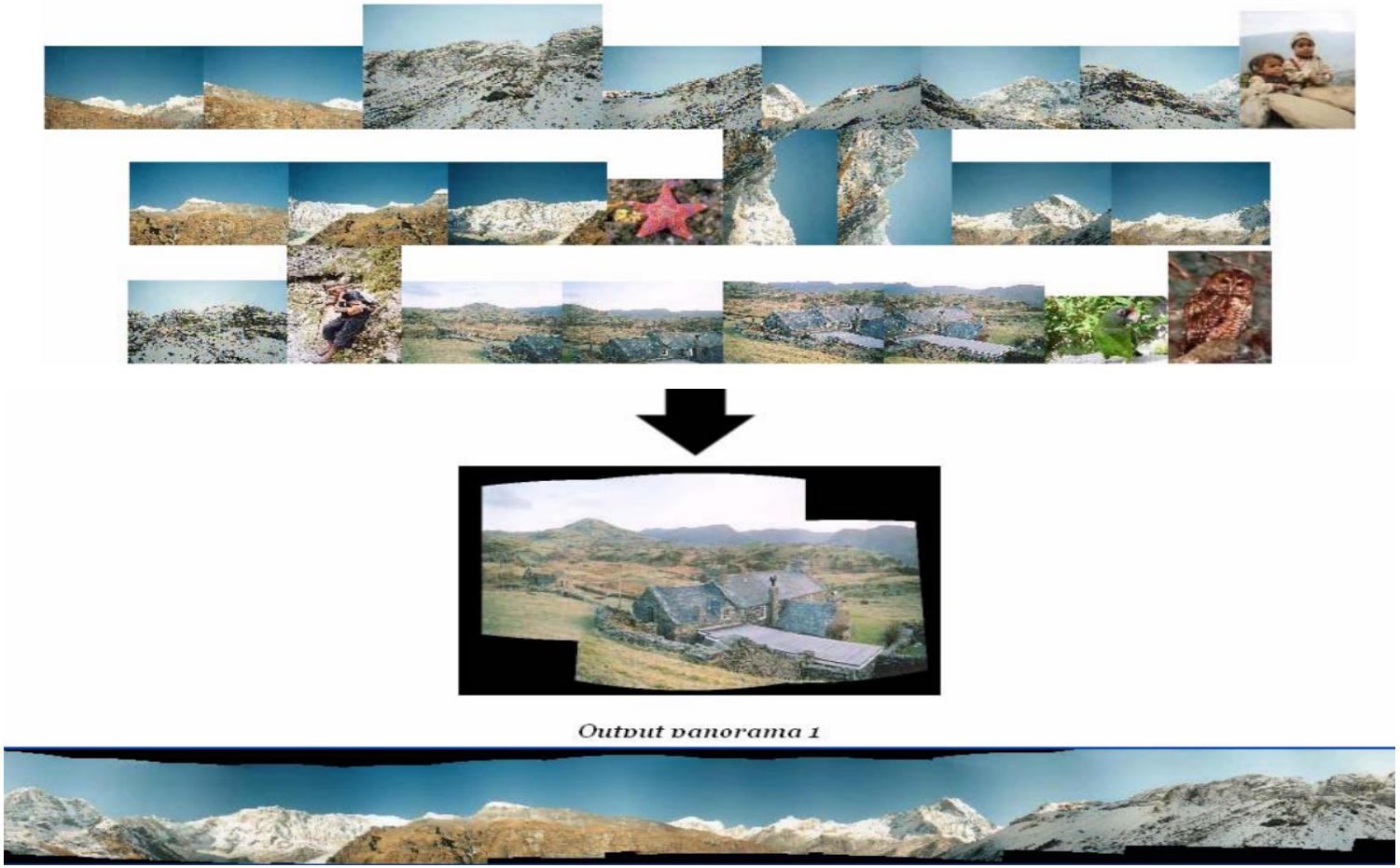
- Error after 400m: 6.5m, 5 degrees (now better)



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Panorama from not-ordered images



Credit: Markus Vincze, Technische Universität Wien

Scale-Invariant Feature Transform - SIFT

Resources

- SIFT patented by David Lowe
- David Lowe's demonstration software
  www.cs.ubc.ca/~lowe/keypoints/
- Autostitch http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html
- Now an app for smart phones or comes with digital camera
- Building Rome in a day [Aggarwal et al., 2009]
  http://grail.cs.washington.edu/rome

Credit: Markus Vincze, Technische Universität Wien

Demos

The following code is taken from the siftFeatureDetection project in the lectures directory of the ACV repository

See:

```
siftFeatureDetection.h
siftFeatureDetectionImplementation.cpp
siftFeatureDetectionApplication.cpp
```

```
/*
  Example use of openCV to find interest point features using the SIFT descriptor (Scale Invariant Feature Transform)
  ------------------------------------------------------------------------------------------------------------

  David Vernon
  27 September 2017
*/

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <ctype.h>
#include <iostream>
#include <string>
#include <conio.h>

//opencv
#include <cv.h>
#include <highgui.h>
#include <opencv2/opencv.hpp>
//#include "opencv2/nonfree/features2d.hpp" // required for SIFT
#include <opencv2/nonfree/nonfree.hpp>      // required for SIFT
```

```
/*
  Example use of openCV to find interest point features using the SIFT descriptor (Scale Invariant Feature Transform)
  ------------------------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  27 September 2017
*/

#include "siftFeatureDetection.h"

void SIFTDetection(char *filename) {

   char*                 input_window_name       = "Input Image";
   char*                 SIFT_window_name        = "SIFT Features";
   Mat                   input_image;
   Mat                   greyscale_image;
   Mat                   sift_features_image;
   Ptr<FeatureDetector>  feature_detector;
   vector<KeyPoint>      SIFT_keypoints;

   input_image = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
   if (input_image.empty()) {
      cout << "can not open " << filename << endl;
      prompt_and_exit(-1);
   }

   printf("Press any key to continue ...\n");
```

```
   /* Create a window for input and display it */
   namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
   imshow(input_window_name, input_image);

   /* Create a window for thresholded image */
   namedWindow(SIFT_window_name, CV_WINDOW_AUTOSIZE );

   if (input_image.type() == CV_8UC3) { // colour image
      cvtColor(input_image, greyscale_image, CV_BGR2GRAY); // SIFT operates on grey-scale
   }
   else {
      greyscale_image = input_image.clone();
   }

   initModule_nonfree(); // required for SIFT

   feature_detector = FeatureDetector::create("SIFT");
   feature_detector->detect(greyscale_image, SIFT_keypoints);
   drawKeypoints(input_image, SIFT_keypoints, sift_features_image, Scalar( 0, 0, 255 ) );

   imshow(SIFT_window_name, sift_features_image);

   do {
      waitKey(30);
   } while (!_kbhit());

   getchar(); // flush the buffer from the keyboard hit

   destroyWindow(input_window_name);
   destroyWindow(SIFT_window_name);
}
```

Demos

The following code is taken from the siftFeatureMatching project in the lectures directory of the ACV repository

See:

```
siftFeatureMatching.h
siftFeatureMatchingImplementation.cpp
siftFeatureMatchingApplication.cpp
```

```
/*
  Example use of openCV to match interest point features using the SIFT descriptor (Scale Invariant Feature Transform)
  ------------------------------------------------------------------------------------------------------------------

  David Vernon
  28 September 2017
*/

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <ctype.h>
#include <iostream>
#include <string>
#include <conio.h>

//opencv
#include <cv.h>
#include <highgui.h>
#include <opencv2/opencv.hpp>
//#include "opencv2/nonfree/features2d.hpp" // required for SIFT
#include <opencv2/nonfree/nonfree.hpp>      // required for SIFT
```

```
/*
  Example use of openCV to match interest point features using the SIFT descriptor (Scale Invariant Feature Transform)
  ------------------------------------------------------------------------------------------------------------------

  David Vernon
  28 September 2017
*/

#include "siftFeatureMatching.h"

void siftMatching(char *filename1, char *filename2) {

   char*                  SIFT_window_name   = "SIFT Descriptor Matches";
   Mat                    input_image1;
   Mat                    input_image2;
   Mat                    greyscale_image1;
   Mat                    greyscale_image2;
   Mat                    descriptors1;
   Mat                    descriptors2;
   Mat                    sift_matches;
   Mat                    temp;
   Mat                    results;
   vector<KeyPoint>       keypoints1;           // keypoints image 1
   vector<KeyPoint>       keypoints2;           // keypoints image 2
   SiftFeatureDetector    sift_detector;        // SIFT features
   SiftDescriptorExtractor sift_extractor;      // SIFT descriptors
   BFMatcher              sift_matcher(NORM_L1); // Brute-force matcher
   vector<DMatch>         matches;              // descriptor matches
```

```
input_image1 = imread(filename1, CV_LOAD_IMAGE_UNCHANGED);
if (input_image1.empty()) {
   cout << "can not open " << filename1 << endl;
   prompt_and_exit(-1);
}

input_image2 = imread(filename2, CV_LOAD_IMAGE_UNCHANGED);
if (input_image2.empty()) {
   cout << "can not open " << filename2 << endl;
   prompt_and_exit(-1);
}

printf("Press any key to continue ...\n");

// Create a window for results image
namedWindow(SIFT_window_name, CV_WINDOW_AUTOSIZE );

if (input_image1.type() == CV_8UC3) { // colour image
   cvtColor(input_image1, greyscale_image1, CV_BGR2GRAY); // SIFT operates on grey-scale
}
else {
   greyscale_image1 = input_image1.clone();
}

if (input_image2.type() == CV_8UC3) { // colour image
   cvtColor(input_image2, greyscale_image2, CV_BGR2GRAY); // SIFT operates on grey-scale
}
else {
   greyscale_image2 = input_image2.clone();
}
```

```
    initModule_nonfree(); // required for SIFT

    /* Find SIFT features */
    sift_detector.detect(greyscale_image1, keypoints1);
    sift_detector.detect(greyscale_image2, keypoints2);

    /* Extract feature descriptors */
    sift_extractor.compute(greyscale_image1, keypoints1, descriptors1);
    sift_extractor.compute(greyscale_image2, keypoints2, descriptors2);

    /* Match descriptors */
    sift_matcher.match(descriptors1, descriptors2, matches);

    /* Display SIFT matches */
    drawMatches(greyscale_image1, keypoints1, greyscale_image2, keypoints2 , matches, sift_matches);
    temp    = JoinImagesHorizontally(input_image1, "", input_image2, "" );
    results = JoinImagesVertically(temp,"",sift_matches,"", 4);
    imshow(SIFT_window_name, results);

    do {
       waitKey(30);
    } while (!_kbhit());

    getchar(); // flush the buffer from the keyboard hit

    destroyWindow(SIFT_window_name);
}
```

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 4.1 Points and Patches

Section 4.1.2 Feature Descriptors

Section 4.1.3 Feature Matching

Section 4.1.4 Feature Tracking

Section 4.1.5 Applications