# 電腦視覺與應用
# Computer Vision and Applications

## Lecture09-Feature detection and matching

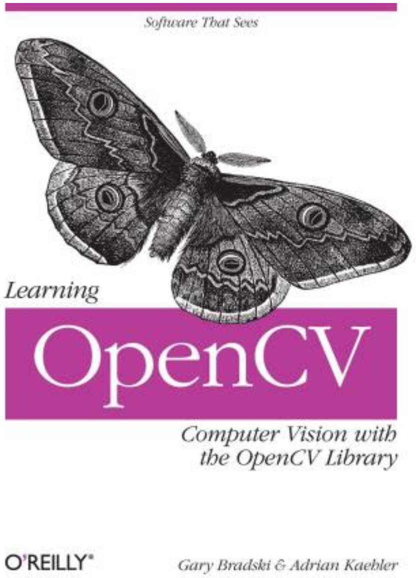
**Tzung-Han Lin**
National Taiwan University of Science and Technology
Graduate Institute of Color and Illumination Technology
e-mail: thl@mail.ntust.edu.tw

TAIWAN TECH National Taiwan University of Science and Technology

CIT 色彩與照明科技研究所 Graduate Institute of Color and Illumination Technology

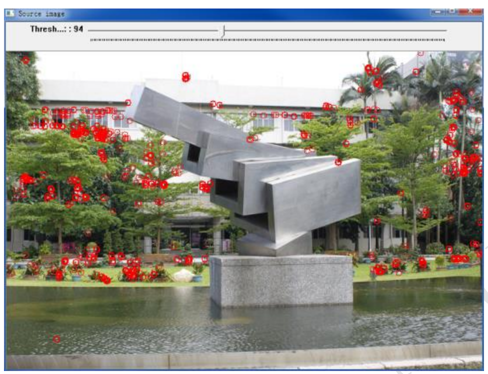
# Feature detection and matching

- Lecture Reference at:
  - G. Bradski and A. Kaehler, *Learning OpenCV Computer Vision with the OpenCV Library*. 2008. (Chapter 10)
  - Intel, OpenCV Reference Manual. 2010. (online or installed with openCV)
  - Select papers.

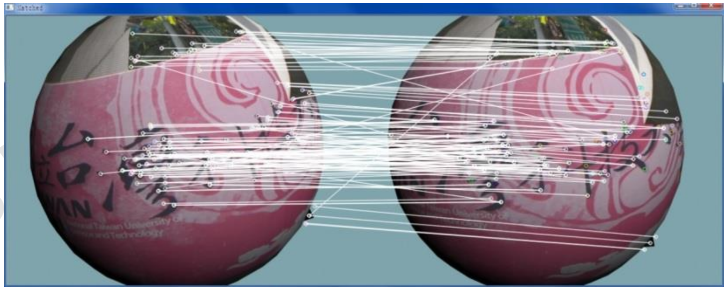


# Feature detection and matching

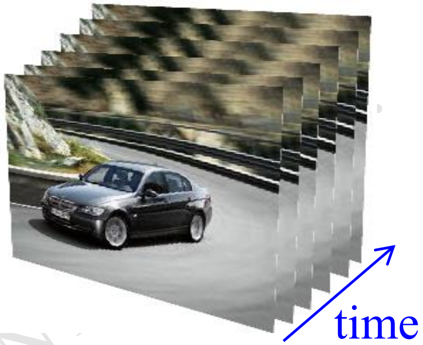
- Brief introduction to
  - Feature Detection→ single frame
  - Feature Matching→ multi images
  - Tracking→ single video



Feature Detection



Feature Matching




Tracking

# Feature detection and matching

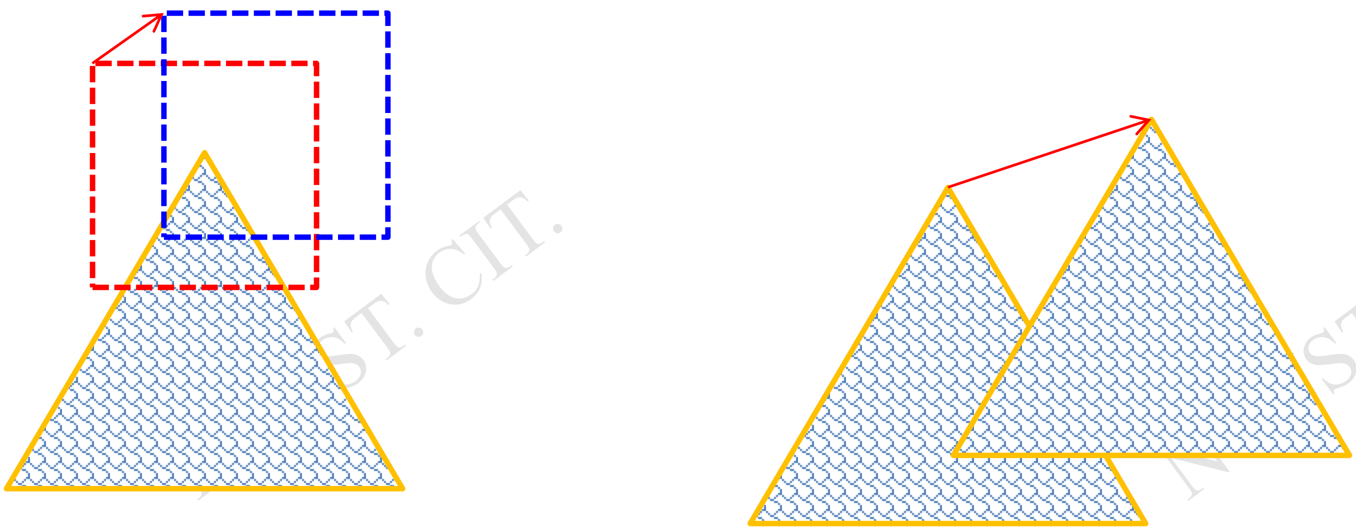
- Feature Detection
  - Harris Corner
  - Tomasi's Good Feature
  - Brown
  - SIFT
  - SURF

---

- High Level feature →machine learning field
  - Face, hand, eye, car, text..
  - codebook, classifier

# Feature detection: Harris corner detector

- Recognize the point by looking through a small window
- Shifting a window in **any direction** should induce **a large change** in intensity



C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147-151.

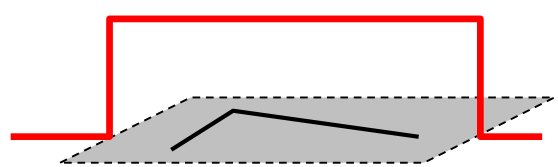
# Feature detection: Harris corner detector

- Estimating "the Change" of intensity when having the shift $[u,v]$

$$E(u,v)=\sum_{x,y} w(x,y)[I(x+u,y+v)-I(x,y)]^2$$

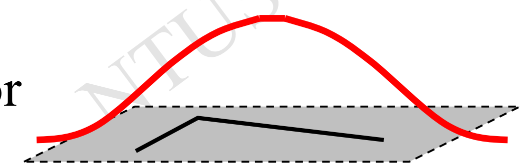
Window function(filter)    Shifted intensity    Intensity

Window function $w(x, y) =$



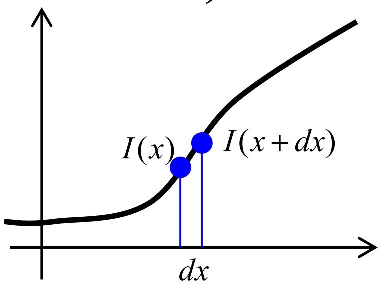
1 in window, 0 outside

or



Gaussian

# Taylor series (Taylor expansion for approximation)




- 1D case

$$I(x+dx) = I(x) + \frac{\partial I(x)}{\partial x} dx$$

First derivatives

$$I(x+\Delta x) = I(x) + I_x(x)\Delta x + \frac{(\Delta x)^2}{2!} I_{xx}(x) + \frac{(\Delta x)^3}{3!} I_{xxx}(x) + \ldots$$

2nd , 3rd derivatives

- 2D case (in image)

$$I(x+u, y+v) = I(x,y) + uI_x(x,y) + vI_y(x,y) + \frac{1}{2!}[u^2 I_{xx}(x,y) + v^2 I_{yy}(x,y) + uvI_{xy}(x,y)] + \ldots.$$

1st partial derivatives

---

- 1st order approximation (for 2D case)

$$I(x+u, y+v) \approx I(x,y) + uI_x(x,y) + vI_y(x,y)$$

# Taylor series (Taylor expansion for approximation)

$$I(x+u, y+v) \approx I(x,y) + uI_x(x,y) + vI_y(x,y)$$

$$\rightarrow I(x+u, y+v) - I(x,y) \approx \underline{uI_x(x,y) + vI_y(x,y)}$$

- Recall Harris corner equation:

$$E(u,v) = \sum_{x,y} w(x,y)[\underline{I(x+u,y+v) - I(x,y)}]^2$$

$$\rightarrow E(u,v) \approx \sum_{x,y} w(x,y)[uI_x(x,y) + vI_y(x,y)]^2$$

$$= \sum_{x,y} w(x,y)[u^2 I_x{}^2 + v^2 I_y{}^2 + 2uvI_x I_y]$$

$$= \sum_{x,y} w(x,y)\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

# Feature detection: Harris corner detector

- For small shifts $[u,v]$ we have a **bilinear** approximation: (from Taylor expansion)

$$E(u,v) \cong [u \quad v]\mathbf{M}\begin{bmatrix} u \\ v \end{bmatrix}$$

- where **M** is a 2×2 matrix computed from image derivatives:

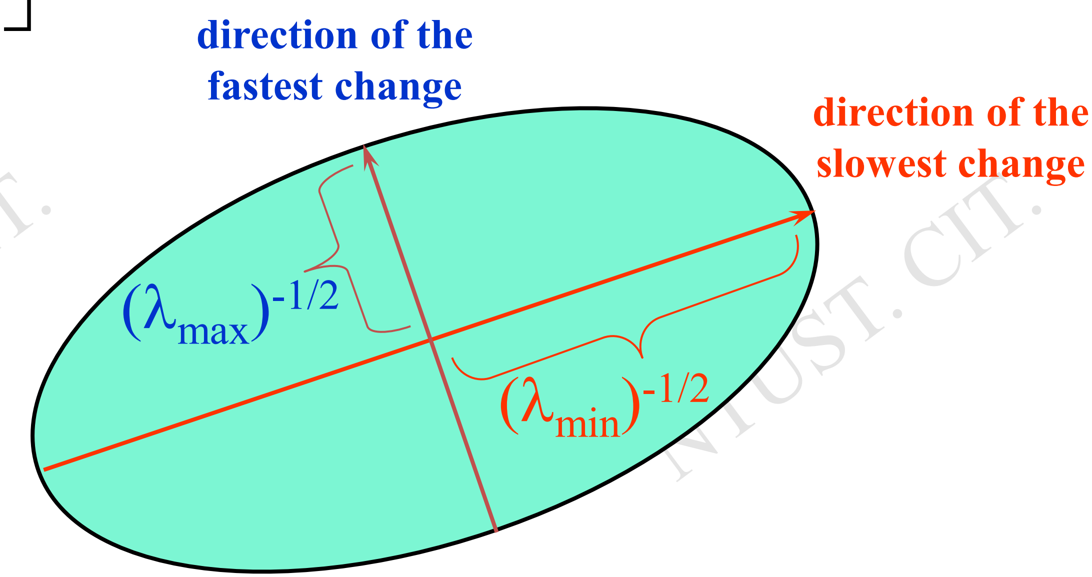
$$\mathbf{M} = \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

# Feature detection: Harris corner detector

- Intensity change in shifting window: **eigenvalue analysis**

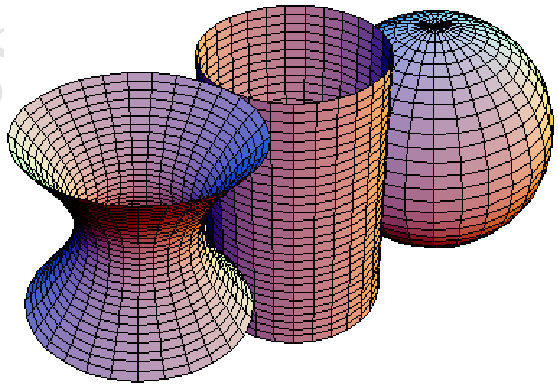
$$E(u,v) \cong \begin{bmatrix} u & v \end{bmatrix} \mathbf{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

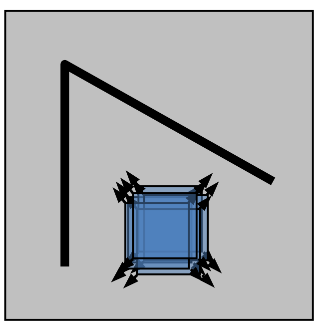
eigenvalue $\lambda_1, \lambda_2$



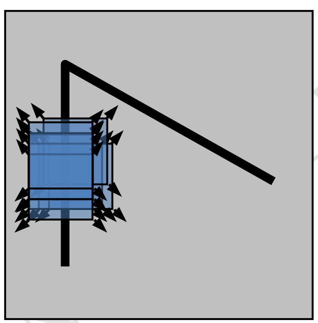
# Feature detection: Harris corner detector



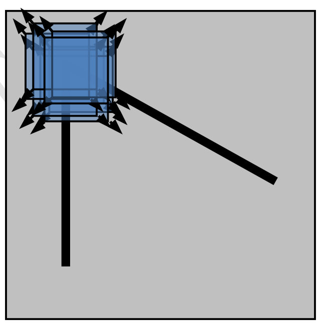




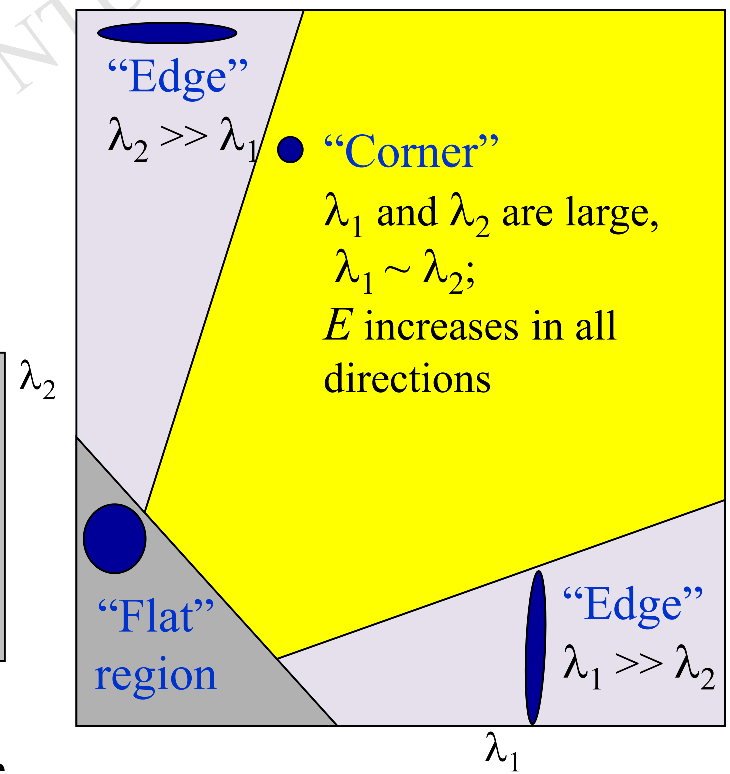
"flat" region:
no change in
all directions



"edge":
no change along
the edge direction



"corner":
significant change
in all directions



# Feature detection: Harris corner detector

- Measure of corner response:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

$$\det(\mathbf{M}) = \lambda_1 \lambda_2$$

$$trace(\mathbf{M}) = (\lambda_1 + \lambda_2)$$

($k$ – empirical constant, $k$ = 0.04~0.06)

---

$\det(\mathbf{M}) = \lambda_1 \lambda_2 \rightarrow$ Gaussion

$trace(\mathbf{M}) = (\lambda_1 + \lambda_2) \rightarrow$ Mean *2

# Harris corners compared to other criterions

$$\mathbf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- Harris corner→ finding max. $R$ $\quad R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$
- Good features to track→ finding the smallest eigenvalue $\min(\lambda_1, \lambda_2)$
- Brown 2005→ finding max. of harmonic mean $\dfrac{\det(\mathbf{M})}{\operatorname{trace}(\mathbf{M})} = \dfrac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$

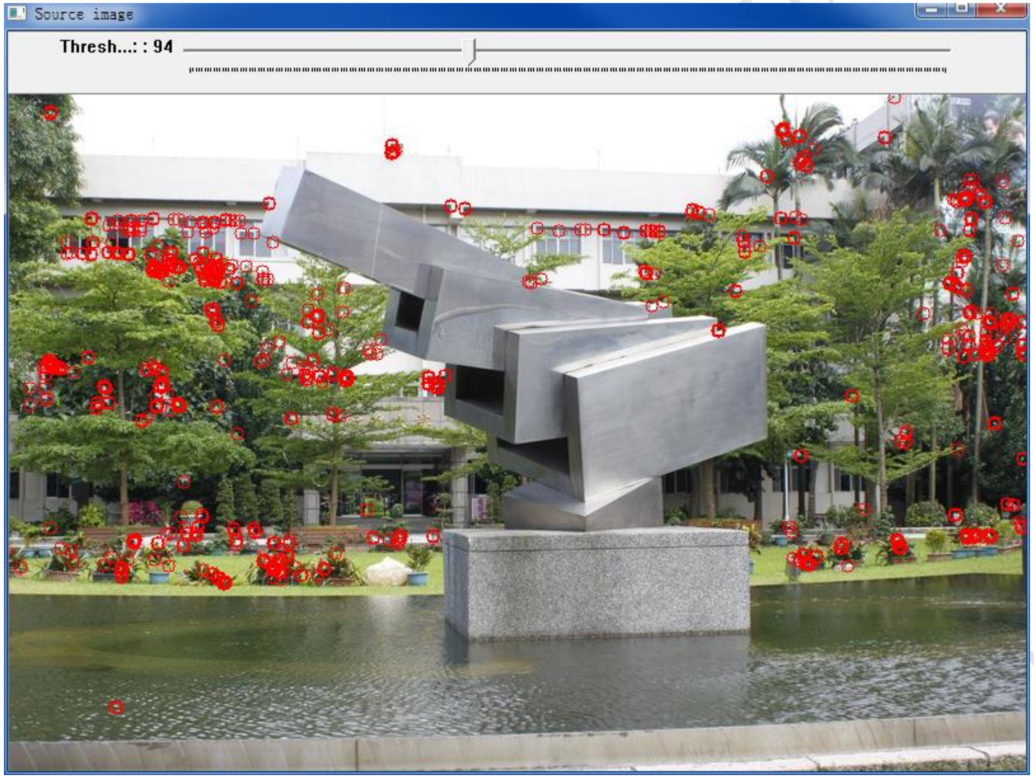
[1] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147-151.
[2] J. Shi and C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, 1994.
[3] M. Brown, R. Szeliski, and S. Winder, "Multi-image matching using multi-scale oriented patches," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, vol. 1, pp. 510-517.

# Corner detection: Summary

- Step 1 : Filter images with Gaussian to reduce noise
- Step 2 : Compute magnitude of $x$ and $y$ gradient at each pixel
- Step 3 : Construct $\mathbf{M}$ matrix and Corner response $R$
- Step 4 : Preserve pixels with $R$ > threshold
- Step 5 : Local maximum suppression

# Feature detection in openCV

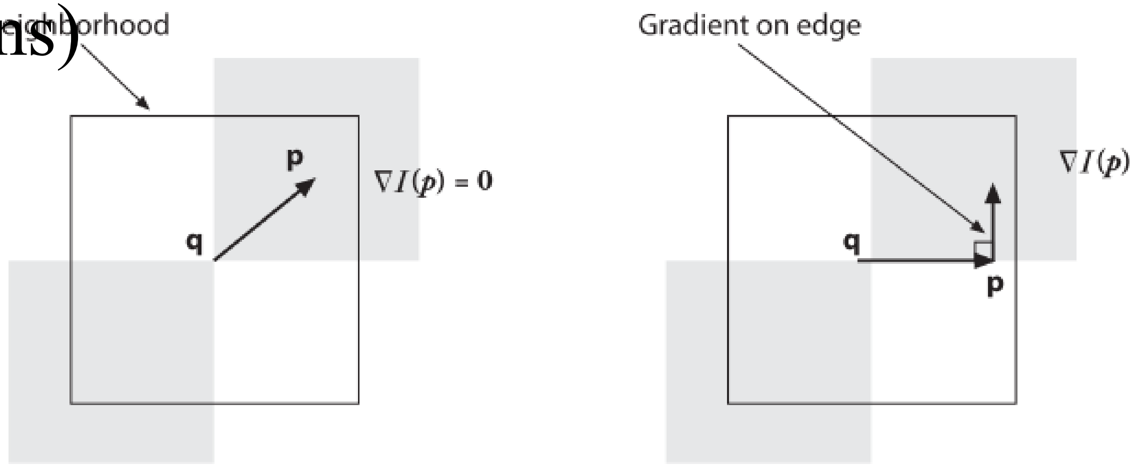
- Example: openCV simple code




(Google "Harris corner openCV")
http://www.opencv.org.cn/opencvdoc/2.3.1/html/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html

# Feature detection in openCV

- Subpixel Corners (for determine precious 2D positions)




(a): p in a region.

(b): p on a region.

In either case the dot product is zero

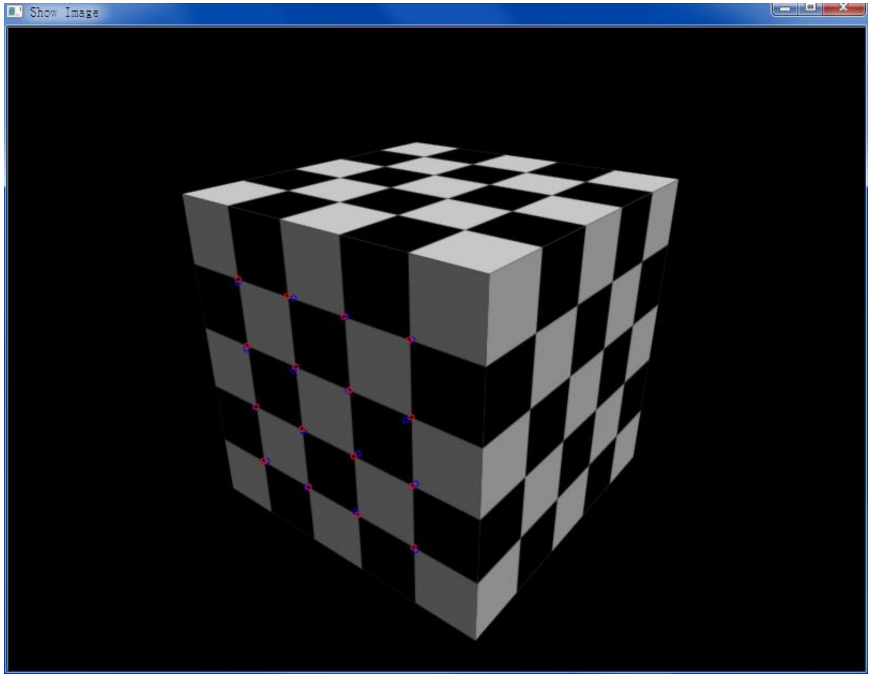
$<\nabla I(p), q-p>=0$

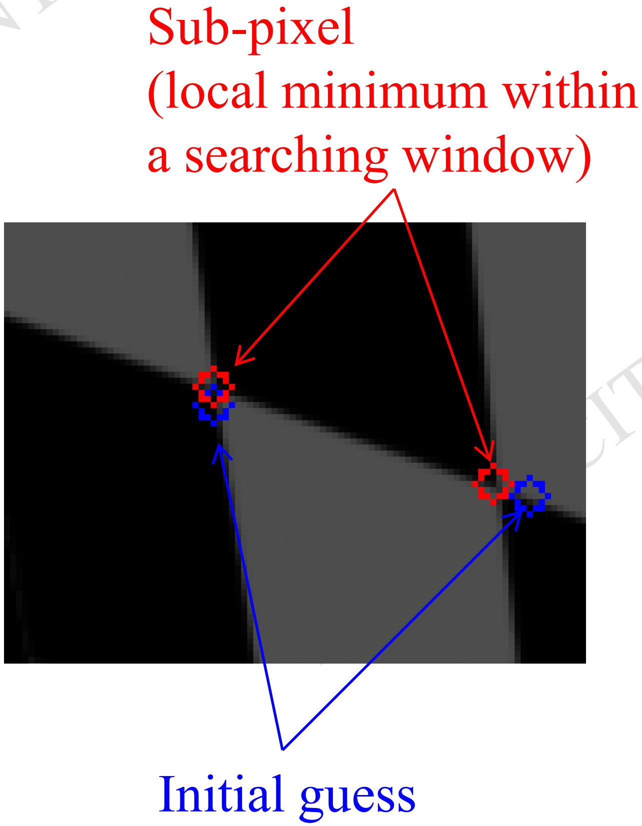
```
//OpenCV sample code
cvFindCornerSubPix(imgB, p, corner_count, cvSize(11,11),cvSize(-1,-1),
cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS , 20, 0.03 ) );
```

D. Chen and G. Zhang, “A new sub-pixel detector for x-corners in camera calibration targets,” in *Central Europe on Computer Graphics Visualization and Computer Vision*, 2005.

# Feature detection in openCV

- Subpixel Corners (for determine precious 2D positions)

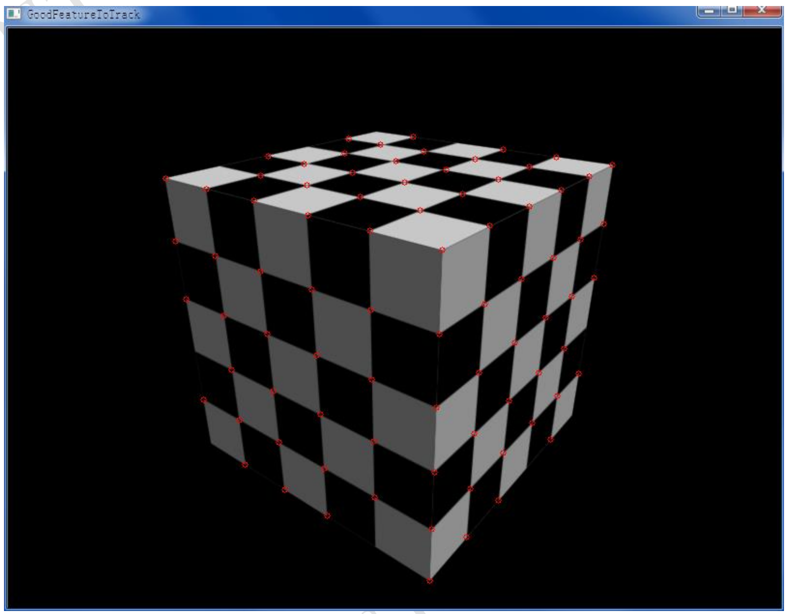


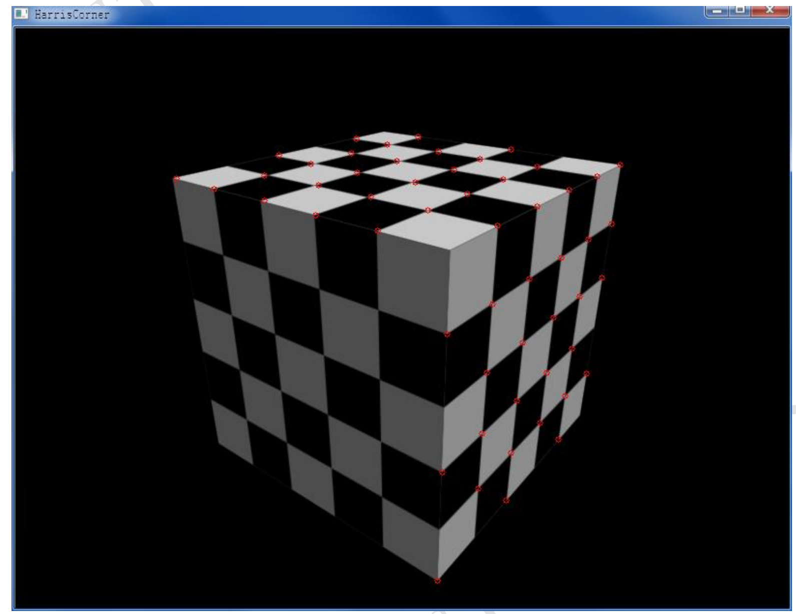

D. Chen and G. Zhang, "A new sub-pixel detector for x-corners in camera calibration targets," in *Central Europe on Computer Graphics Visualization and Computer Vision*, 2005.

# Feature detection in openCV

- Subpixel Corners (coupled with Harris Corner)



cvGoodFeaturesToTrack
+cvFindCornerSubPix



cvGoodFeaturesToTrack
(Harris criterion)
+cvFindCornerSubPix

# Feature detection in openCV

- Subpixel Corners (coupled with Harris Corner)

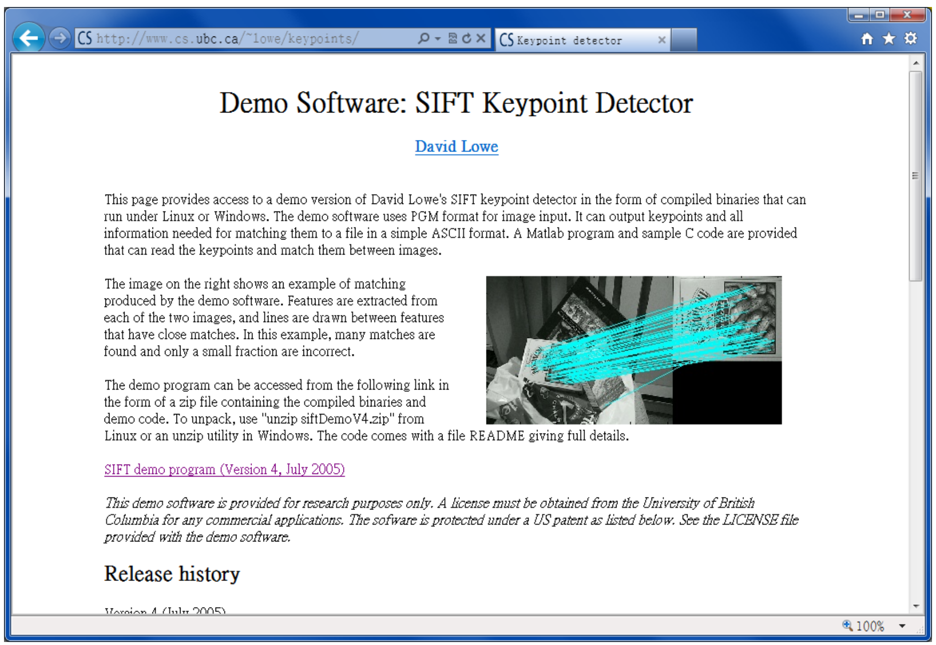



cvGoodFeaturesToTrack
+cvFindCornerSubPix


cvGoodFeaturesToTrack
(Harris criterion)
+cvFindCornerSubPix

# Feature detection

- SIFT: Scale Invariant Feature Transform
- Other type: PCA-SIFT, GLOH




Hessian Matrix

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

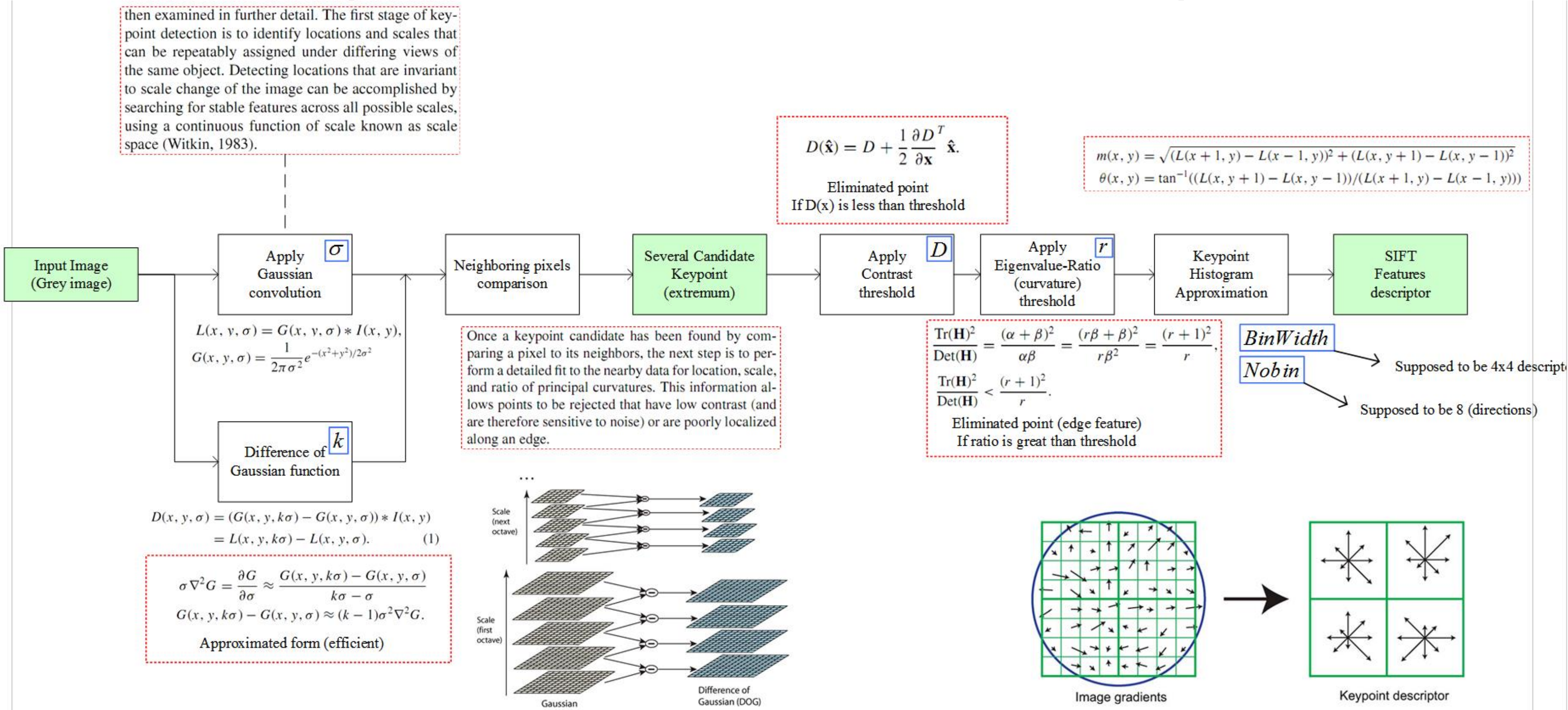
Reject when

$$\frac{[trace(\mathbf{H})]^2}{\det(\mathbf{H})} > threshold$$

Demo code: http://www.cs.ubc.ca/~lowe/keypoints/

D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, Nov. 2004.

# Summary of SIFT

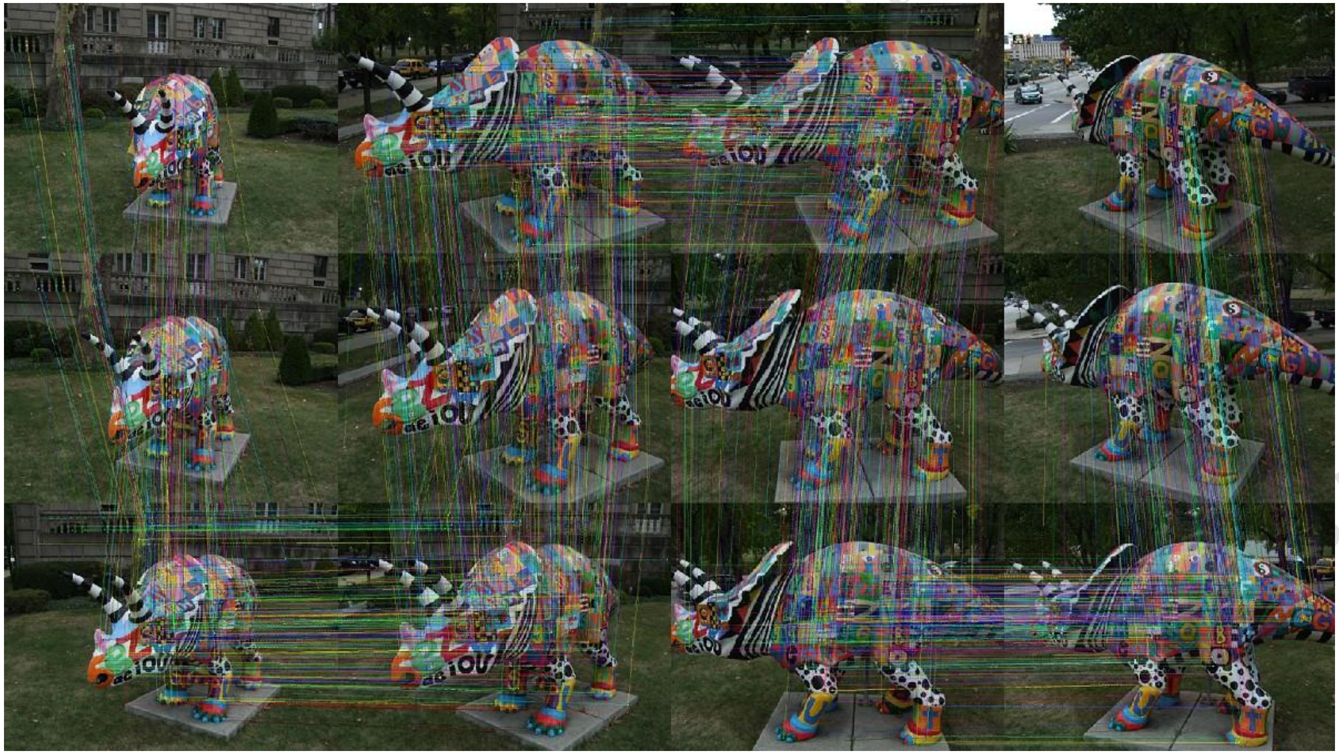



Optimized / Suggestion parameter

tave of scale space. The image doubling increases the number of stable keypoints by almost a factor of 4, but no significant further improvements were found with a larger expansion factor. $imgDBL$

# Feature detection

- SIFT: example



# Feature detection

- SURF: Speeded Up Robust Features


Hessian Matrix

$$\mathbf{H}(x,\sigma)=\begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix}$$

**//Sample code in openCV**

```
cv::SurfFeatureDetector surf(2500);
surf.detect(image1, keypoints1);
drawKeypoints(image1, keypoints1, outImg1, Scalar(255,255,255),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

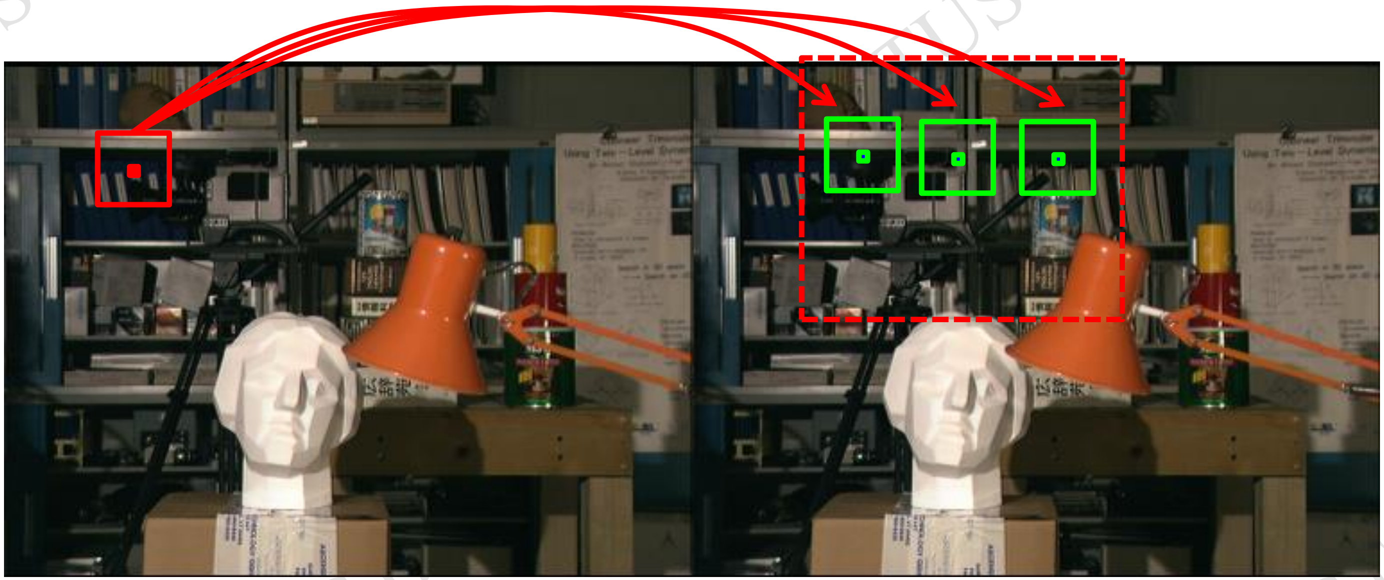
H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision*, 2006, pp. 404-417.

# Feature matching

- Comparison of patches (block matching)
  - brute force
  - strategy
  - transform → frequency domain FFT
  - GPU (acceleration issue)
- Comparison of feature descriptor (keypoint)

# Feature matching

- Template matching by a cropped patch



For estimating the disparity of ONE pixel (in the left image)
Transfer the image into grey level, then,

1. Extend(crop) a window on the left image
2. Searching from the same pixel position, compare the difference in-between the upper / lower bound region
3. Pick out the most similar region as the matching position.

# Feature matching

- Matching Criteria – Difference
  - Common matching criteria
  - **SSD - Sum of Squared Differences** (min. for match)

$$d(p_1, p_2) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} \left( I_1(x_1 + i, y_1 + j) - I_2(x_2 + i, y_2 + j) \right)^2$$

$k$ - the size of the window.

$p_1 = (x_1, y_1)$ is the center of the window in $I_1$.

$p_2 = (x_2, y_2)$ is the center of the window in $I_2$.

# Feature matching

- Matching Criteria – Difference
  - Common matching criteria
  - **SAD** - **Sum of Absolute Differences** (min. for match)

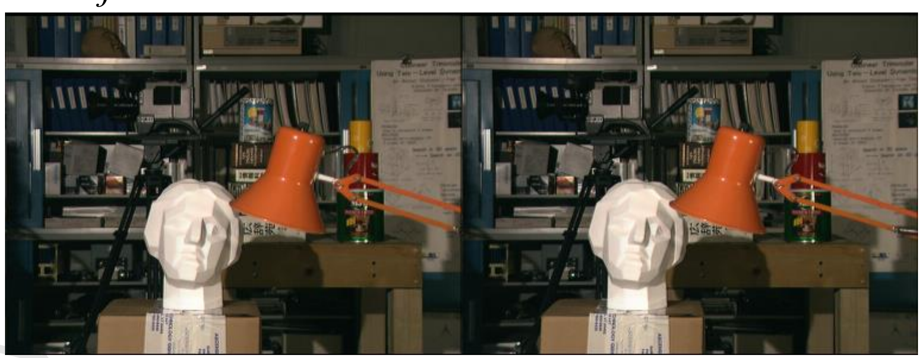
(**MAD** - **M**inimum **A**bsolute **D**ifference)

$$d(p_1, p_2) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} \left| I_1(x_1 + i, y_1 + j) - I_2(x_2 + i, y_2 + j) \right|$$

# Feature matching

- Matching Criteria – Difference
  - Common matching criteria
  - Cross correlation (the max. value for the matched point)

$$d(p_1, p_2) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} I_1(x_1 + i, y_1 + j) \cdot I_2(x_2 + i, y_2 + j)$$



- In **SSD**, **SAD** and **Cross Correlation**, the brightness is assumed to be similar.

# Feature matching

- Matching Criteria – Difference
  - NCC (Normalized Cross Correlation)

- When ordering the pixels in the windows in vectors $v_1$, $v_2$:

$$SSD(\vec{v}_1,\vec{v}_2) = \sum_i (\vec{v}_1(i) - \vec{v}_2(i))^2 = \|\vec{v}_1 - \vec{v}_2\|^2 \longleftarrow \text{Squared vector norm}$$

$$Corr(\vec{v}_1,\vec{v}_2) = \sum_i \vec{v}_1(i) \cdot \vec{v}_2(i) = \langle \vec{v}_1, \vec{v}_2 \rangle \longleftarrow \text{Inner product}$$

- Normalized Cross Correlation is:

$$NCC(\vec{v}_1,\vec{v}_2) = \frac{\sum_i \vec{v}_1(i) \cdot \vec{v}_2(i)}{\sqrt{\sum_i \vec{v}_1(i) \cdot \vec{v}_1(i)} \sqrt{\sum_i \vec{v}_2(i) \cdot \vec{v}_2(i)}} = \frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{\|\vec{v}_1\| \|\vec{v}_2\|}$$

# Feature matching

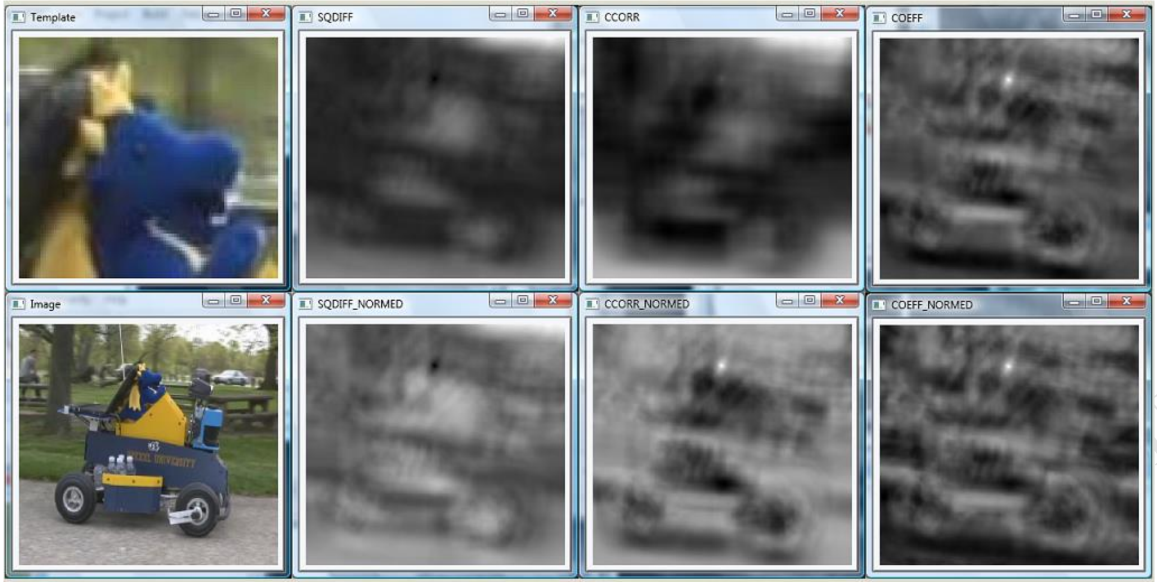
- Limitations of Matching
  - Accuracy:
    - A pixel is always matched to integer location on the grid. The image motion is usually not integer.
  - Neighborhood/Scene constraints:
    - High level knowledge about the scene/camera may help in limiting the search, and reducing errors.

# Feature matching

- Block Matching in openCV

openCV function: cvMatchTemplate

```
void cvMatchTemplate(const CvArr* image,
                     const CvArr* templ,
                     CvArr* result,
                     int method );
```




Online resource
http://dasl.mem.drexel.edu/~noahKuntz/openCVTut6.html

# Feature matching

- Block Matching in openCV
  - method=CV_TM_SQDIFF
    → sum of square difference
  - method=CV_TM_SQDIFF NORMED
    → normalized sum of square difference
  - method=CV_TM_CCORR
    → cross correlation
  - method=CV_TM_CCORR_NORMED
    → normalized cross correlation
  - method=CV_TM_CCOEFF
    → correlation coefficient
  - method=CV_TM_CCOEFF_NORMED
    → fast normalized correlation coefficient (FFT)

http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html#tex2html1

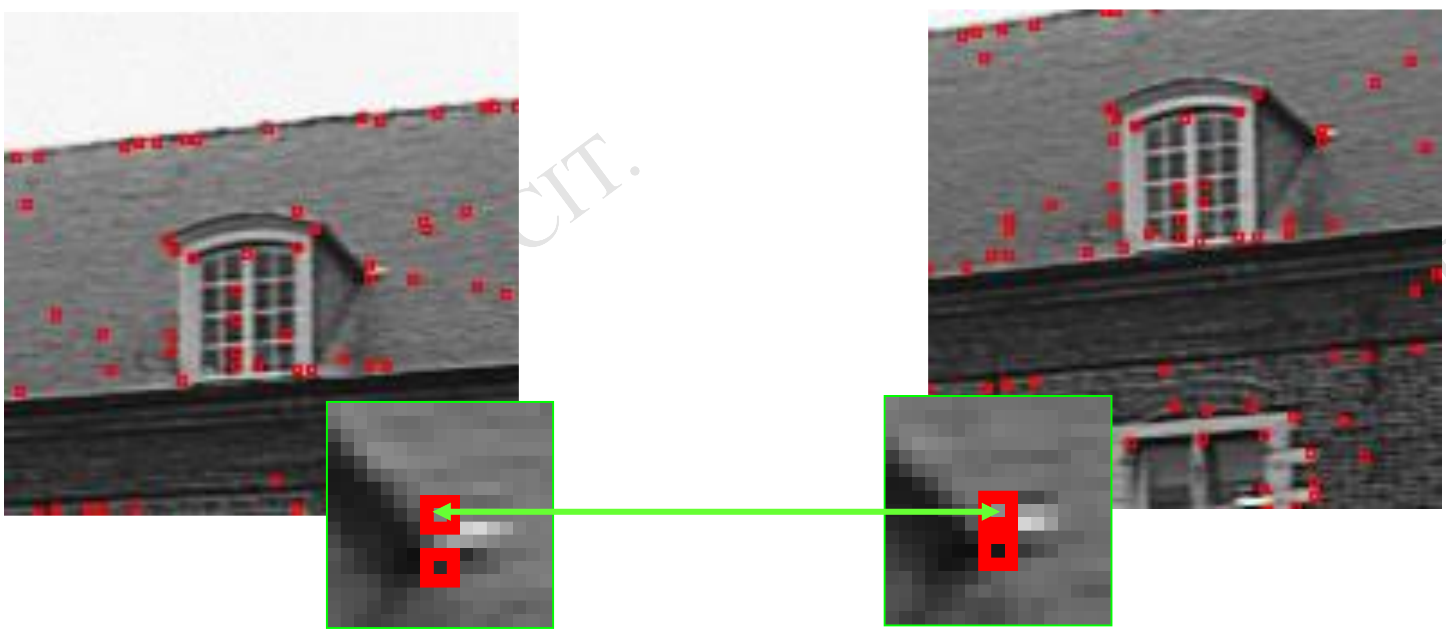
# Feature matching

- Stereo-Block matching example


Video source from:
http://www.youtube.com/watch?v=FX7AMktf24E

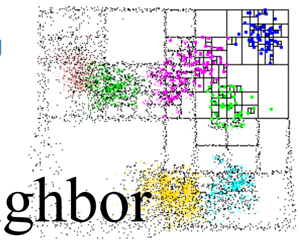
# Feature matching

- Matching for known features
  - for each corner in image 1 find the corner in image 2 that is most similar (using **SSD** or **NCC**) and vice-versa
  - Only compare geometrically compatible points
  - Keep mutual best matches



# Feature matching

**ANN: A Library for Approximate Nearest Neighbor Searching**
David M. Mount and Sunil Arya
Version 1.1.2
Release Date: Jan 27, 2010



- Fast Searching Issue
  - ANN or FLANN (Fast Approximate Nearest Neighbor Search), KD-Tree

**What is ANN?**

ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.

ANN: http://www.cs.umd.edu/~mount/ANN/


Harris + FLANN

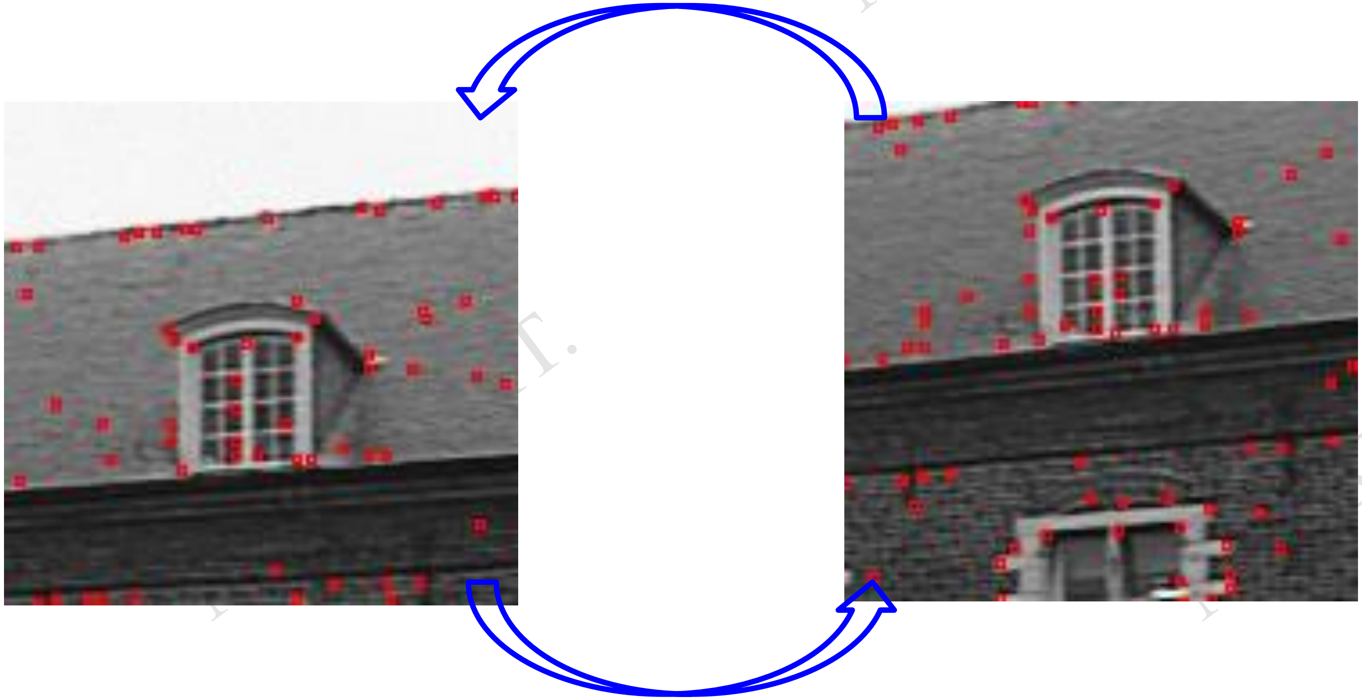
- openCV example please See: Intel, "The OpenCV Tutorials," 2011. Chapter 6



[1] K. Hajebi, Y. Abbasi-yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *International Joint Conference on Artificial Intelligence*, 2009, no. C, pp. 1312-1317.
[2] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Applications*, 2009, pp. 331-340.

# Feature matching

- Feature → epipolar geometry(**F** matrix) ?
  epipolar geometry → Feature ?



# Feature matching

- Example: SIFT→RANSAC

(a) Image 1 (b) Image 2

(c) SIFT matches 1 (d) SIFT matches 2

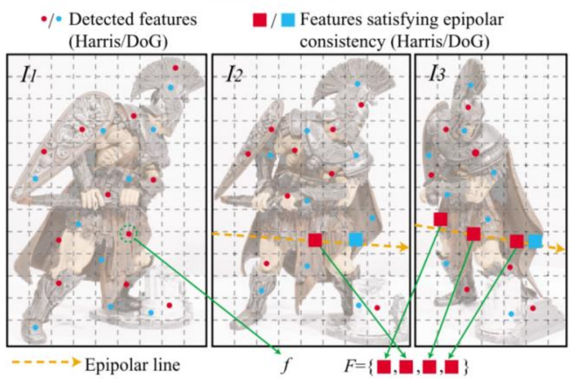
(e) RANSAC inliers 1 (f) RANSAC inliers 2



M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59-73, Dec. 2006.

# Feature matching

- Example:SIFT→RANSAC(epipolar geometry) →Harris corner detection→….






Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, vol. 32, no. 8, pp. 1362-76.

# Feature tracking

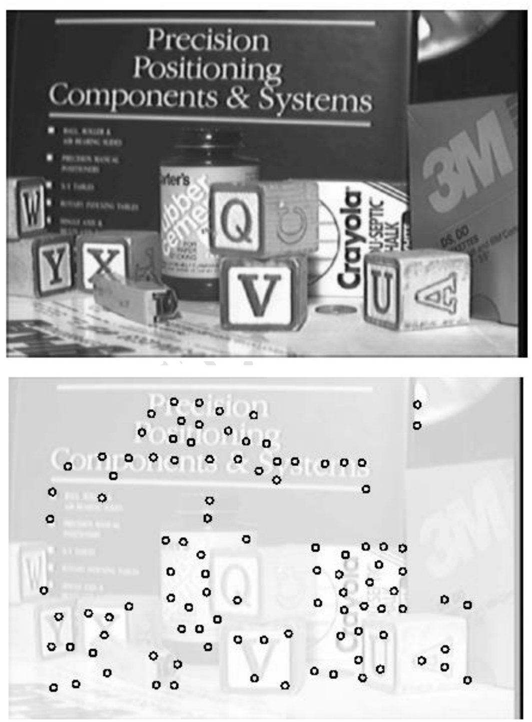
- Tracking in "Single Video Stream"
  - Optical flow
    - by block matching(template matching)
    - by Horn-Schunck
    - by Lucas-Kanade
    - by KLT
  - Kalman filter

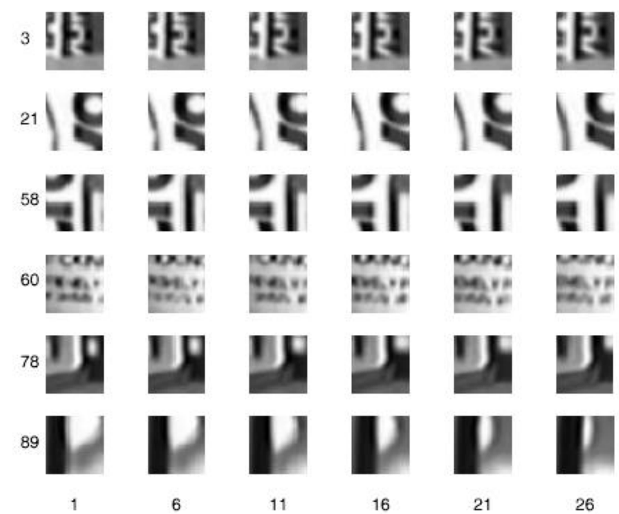
# Feature tracking

- Identify features and track them over video
  - Small difference between frames
  - potential large difference overall
- Standard approach:
  - KLT (Kanade-Lukas-Tomasi)

# Feature tracking

- Good features to keep tracking
  - Perform affine alignment between first and last frame. Stop tracking features with too large errors

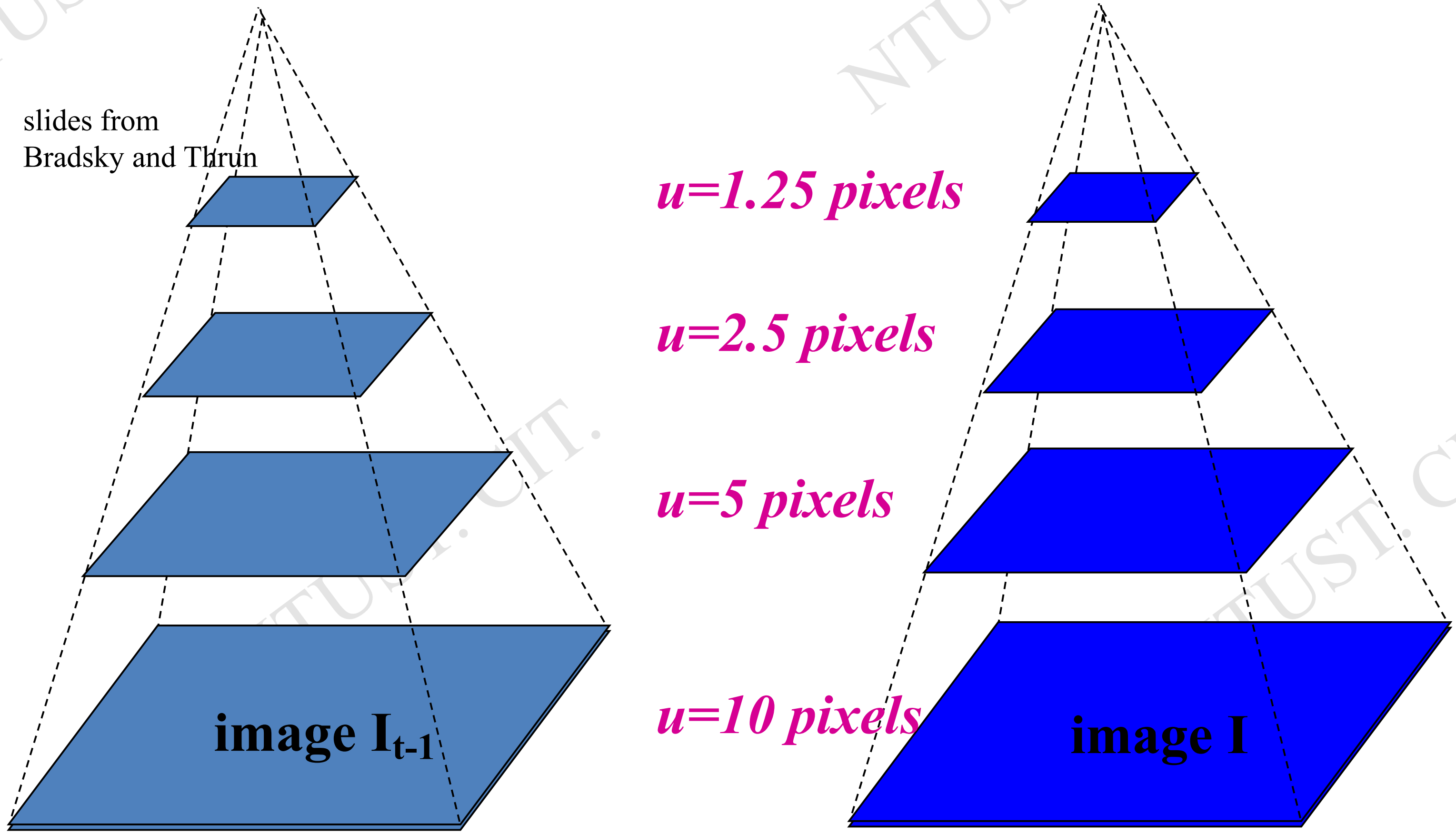


# Feature tracking

- Coarse-to-fine optical flow estimation

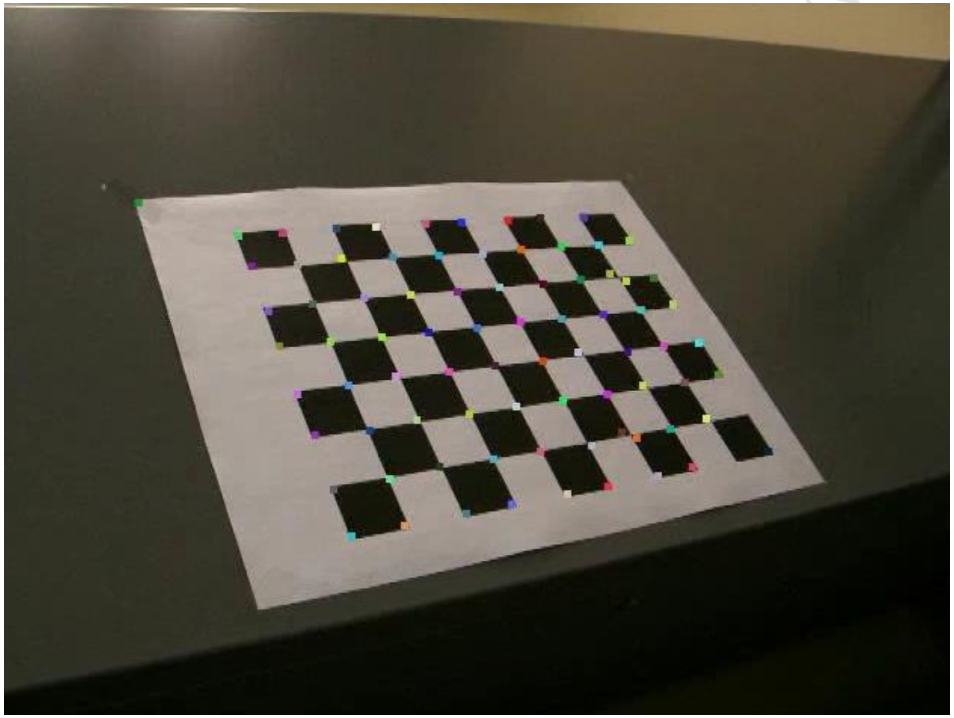
slides from
Bradsky and Thrun




Gaussian pyramid of image $I_{t-1}$ Gaussian pyramid of image I

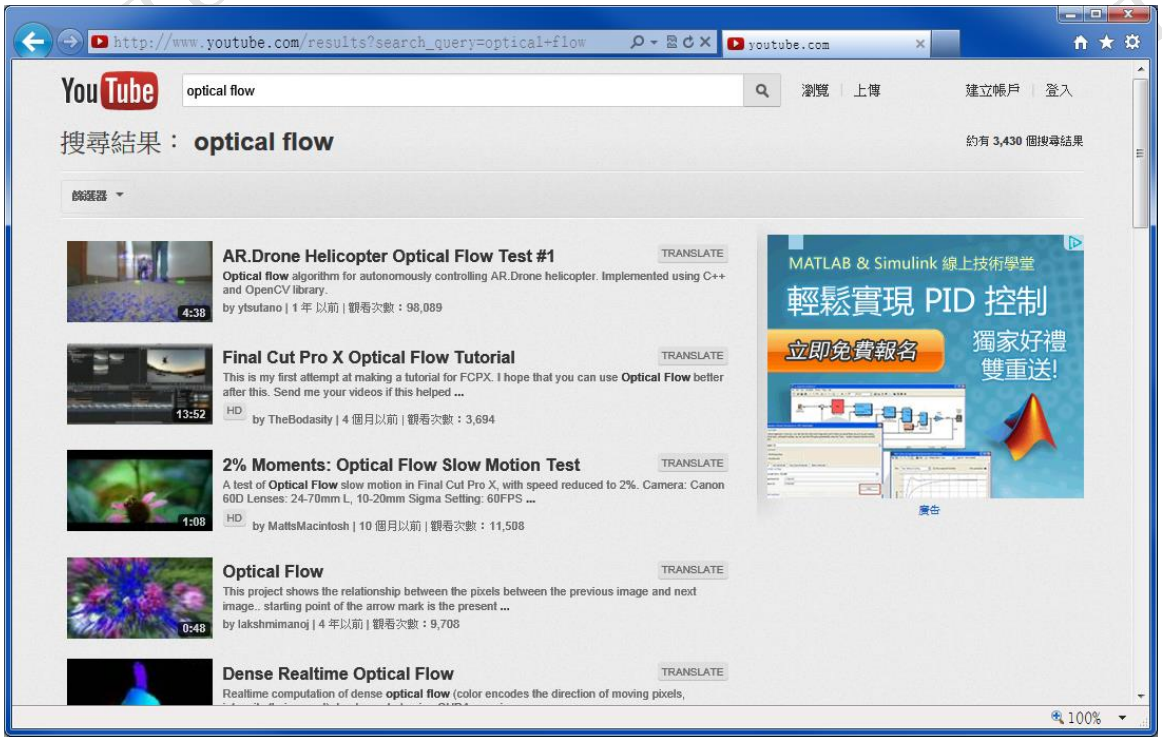
# Feature tracking

- KLT tracking example



# Feature tracking

- Optical flow, example




Optical flow function in openCV

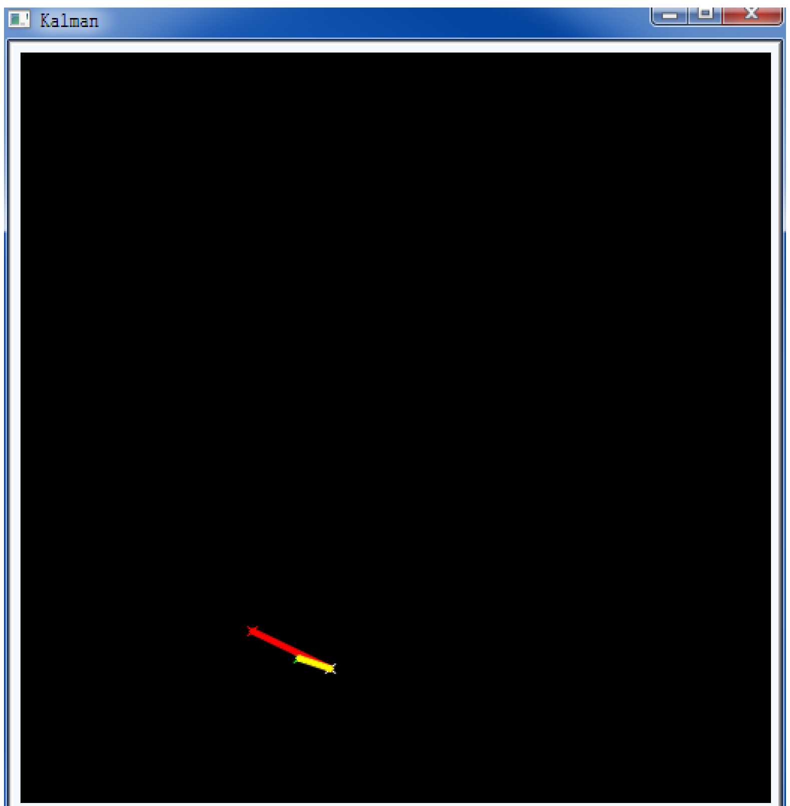
cvCalcOpticalFlowBM
→using block matching

cvCalcOpticalFlowHS
→using Horn and Schunck

cvCalcOpticalFlowLK
using Lucas-Kanade
cvCalcOpticalFlowPyrLK
→using Lucas-Kanade (LoD)

# Feature tracking

- Kalman filter, example
  - Pre-state, Post-state, Predict state, Update state




Kalman filter in openCV

cvKalman
cvCreateKalman
cvKalmanPredict
cvKalmanCorrect

# Kalman filter

- Kalman filter
  - Formula

$$\underbrace{y_k}_{\text{observed}} = \underbrace{a_k}_{\text{gain}} \underbrace{x_k}_{\text{to be estimated}} + \underbrace{n_k}_{\text{noise}}$$

  - Error function

$$f(e_k) = f(x_k - \hat{x}_k) = (x_k - \hat{x}_k)^2$$

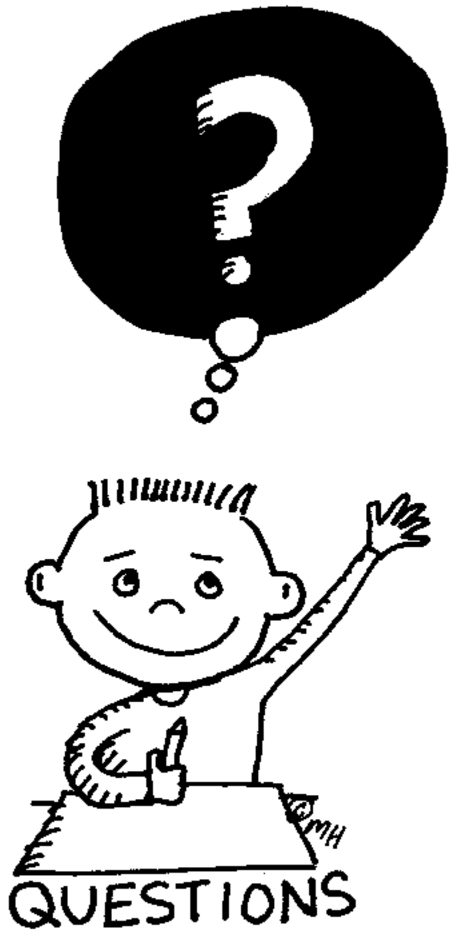





Figure from : http://www.gabankruptcylawyersnetwork.com/files/2014/08/question.gif