

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

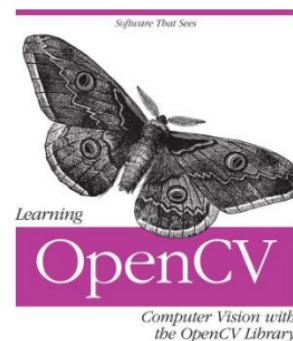
e-mail: thl@mail.ntust.edu.tw





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.



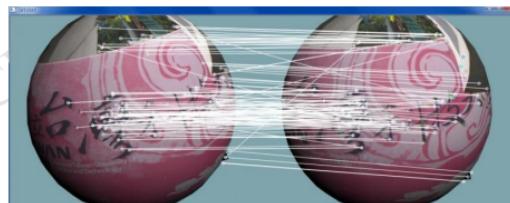


Feature detection and matching

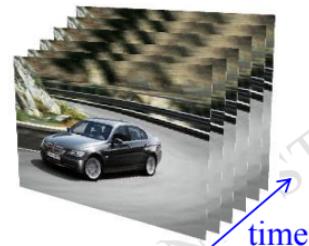
- Brief introduction to
 - Feature Detection → single frame
 - Feature Matching → multi images
 - Tracking → single video



Feature Detection



Feature Matching



Tracking



Feature detection and matching

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

- SIFT
- SURF

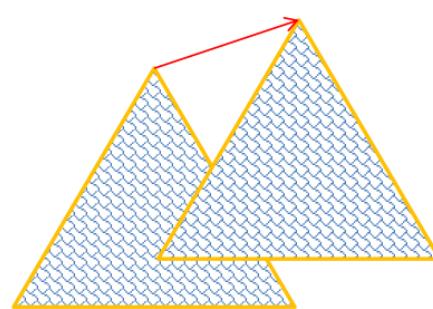
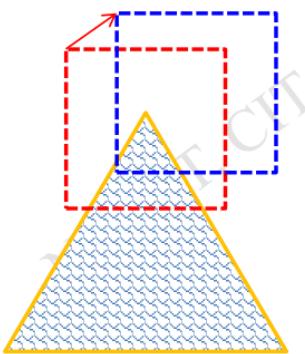


- High Level feature → machine/deep learning field (recognition)
 - Face, hand, eye, car, text..
 - codebook, classifier



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





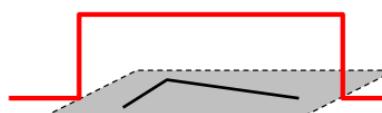
Feature detection: Harris corner detector

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

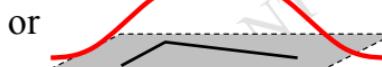
$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2$$

Diagram illustrating the Harris corner detector formula. It shows a window function (filter) $w(x, y)$ applied to the original intensity $I(x, y)$ and the shifted intensity $I(x + u, y + v)$. The result is squared and summed to calculate the Harris response $E(u, v)$.

Window function $w(x, y) =$



1 in window, 0 outside



Gaussian

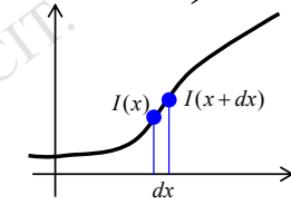


Taylor series (Taylor expansion for approximation)

■ 1D case

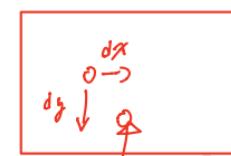
$$I(x+dx) = I(x) + \frac{\partial I(x)}{\partial x} dx$$

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



■ 2D case (in image)

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



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

■ 1st order approximation (for 2D case)



Taylor series (Taylor expansion for approximation)

$$\begin{aligned} I(x+u, y+v) &\approx I(x, y) + uI_x(x, y) + vI_y(x, y) \\ \rightarrow I(x+u, y+v) - I(x, y) &\approx \underline{uI_x(x, y)} + \underline{vI_y(x, y)} \quad + (\rightarrow) \end{aligned}$$

- Recall Harris corner equation:

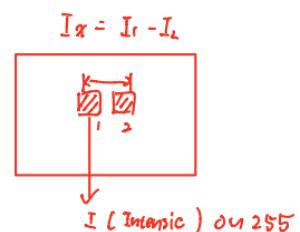
$$\begin{aligned} E(u, v) &= \sum_{x,y} w(x, y) [\underline{I(x+u, y+v)} - \underline{I(x, y)}]^2 \\ \rightarrow E(u, v) &\approx \sum_{x,y} w(x, y) [uI_x(x, y) + vI_y(x, y)]^2 \\ &= \sum_{x,y} w(x, y) [u^2 I_x^2 + v^2 I_y^2 + 2uv I_x I_y] \\ &= \sum_{x,y} w(x, y) [u \quad v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$



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 \mathbf{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

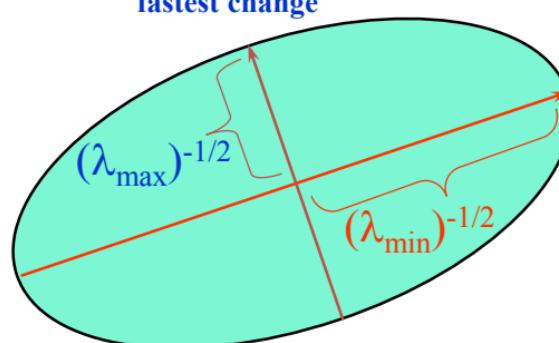
- Intensity change in shifting window: eigenvalue analysis

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

eigenvalue λ_1, λ_2

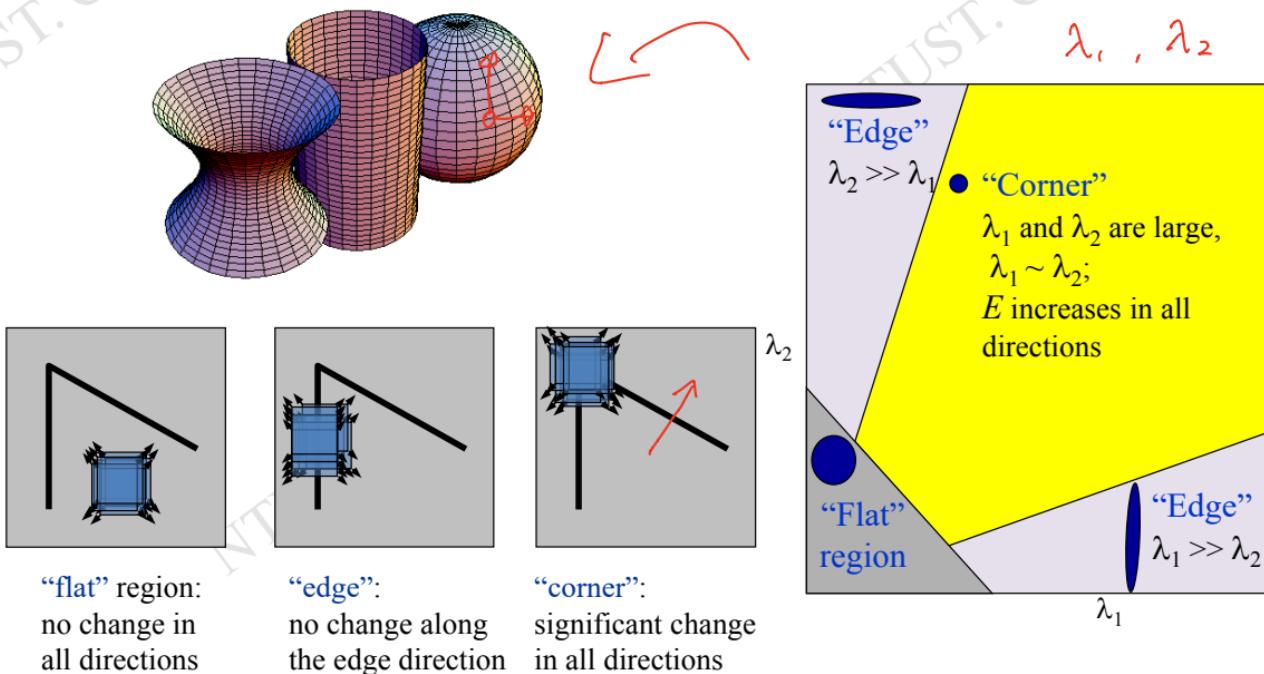
direction of the
fastest change

direction of the
slowest change





Feature detection: Harris corner detector





Feature detection: Harris corner detector

- Measure of corner response:

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

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

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

(k – empirical constant, $k = 0.04\sim0.06$)

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

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



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

[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 **M** matrix and Corner response **R**

Step 4 : Preserve pixels with $\mathbf{R} >$ threshold

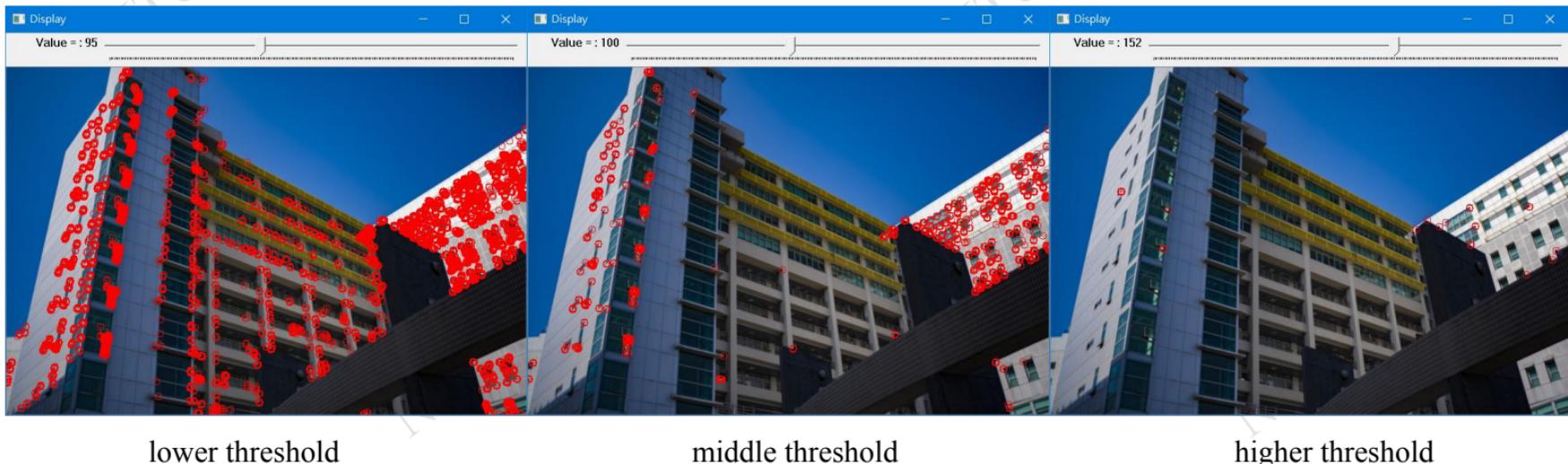
Step 5 : Local maximum suppression

$$I_x = \frac{I_i - I_j}{1.8}$$



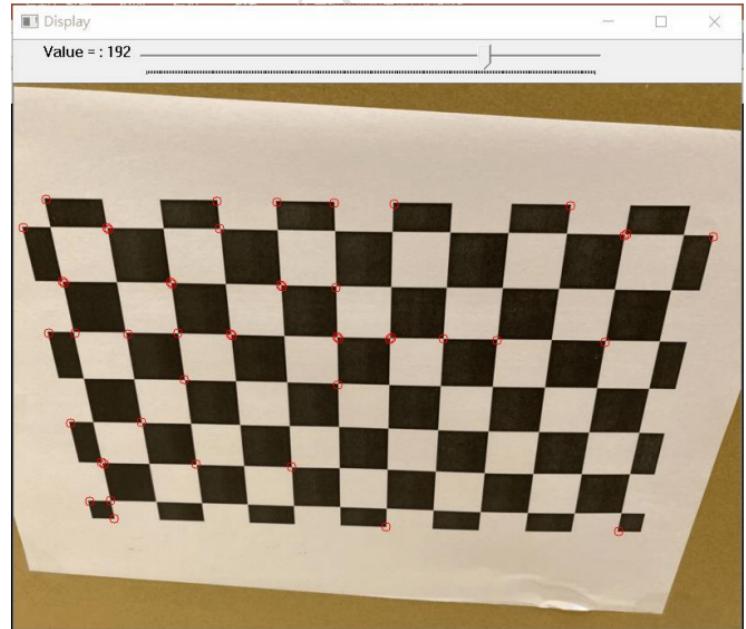
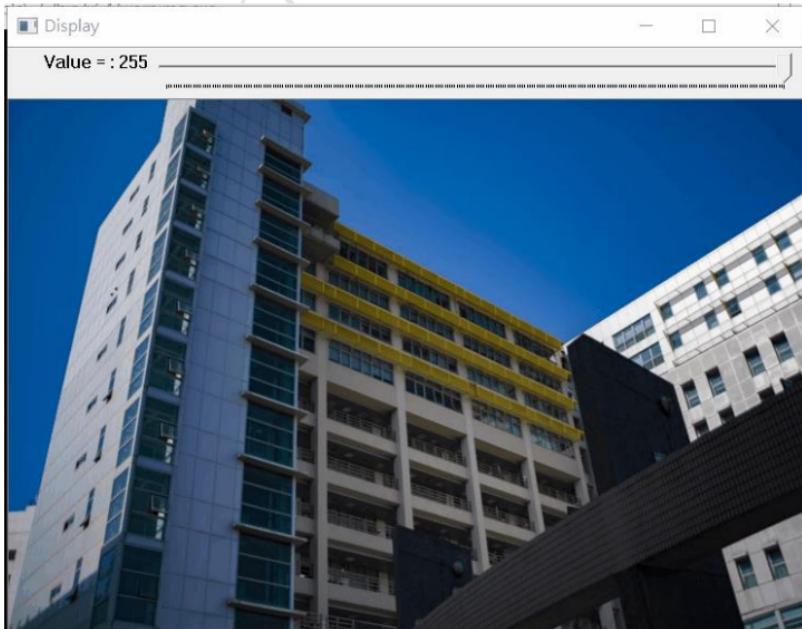
Feature detection in openCV

- Example: openCV simple code





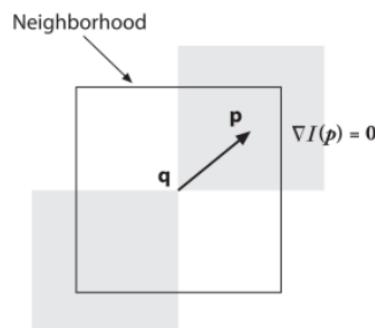
Harris corner in openCV



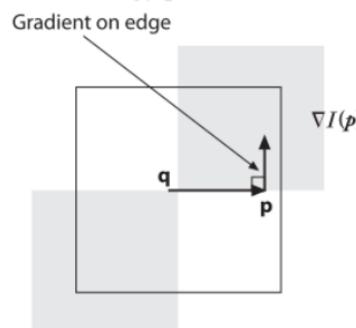


Feature detection in openCV

- 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

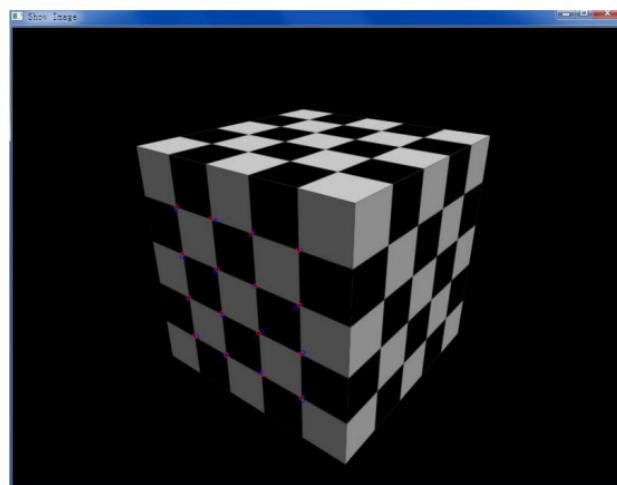
$$\langle \nabla I(p), q - p \rangle = 0$$

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

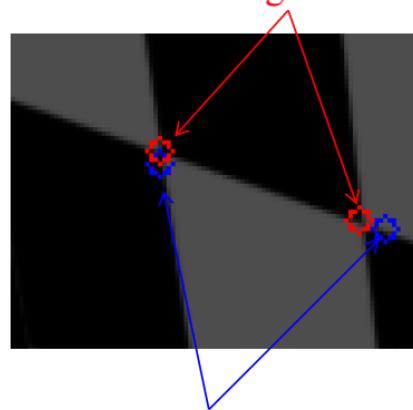


Feature detection in openCV

- Subpixel Corners (for determine precious 2D positions)



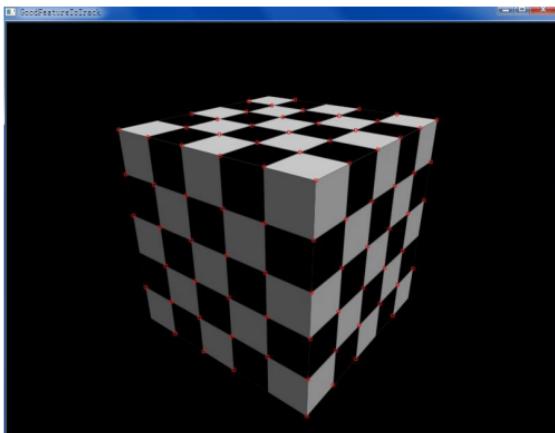
Sub-pixel
(local minimum within
a searching window)



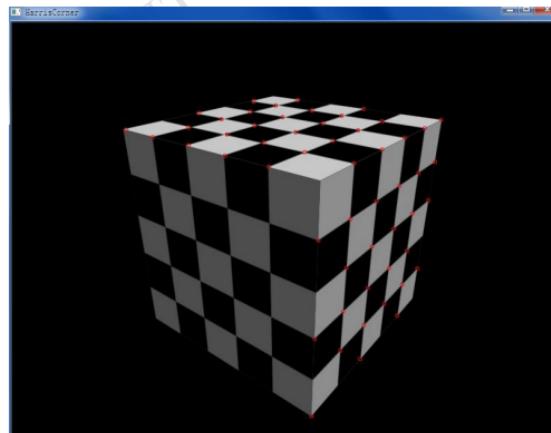


Feature detection in openCV

- Subpixel Corners (coupled with Harris Corner)



cvGoodFeaturesToTrack
+cvFindCornerSubPix



cvGoodFeaturesToTrack
(Harris criterion)
+cvFindCornerSubPix

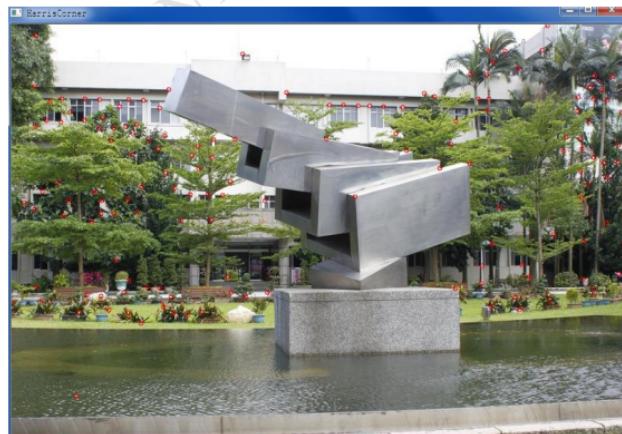


Feature detection in openCV

- Subpixel Corners (coupled with Harris Corner)



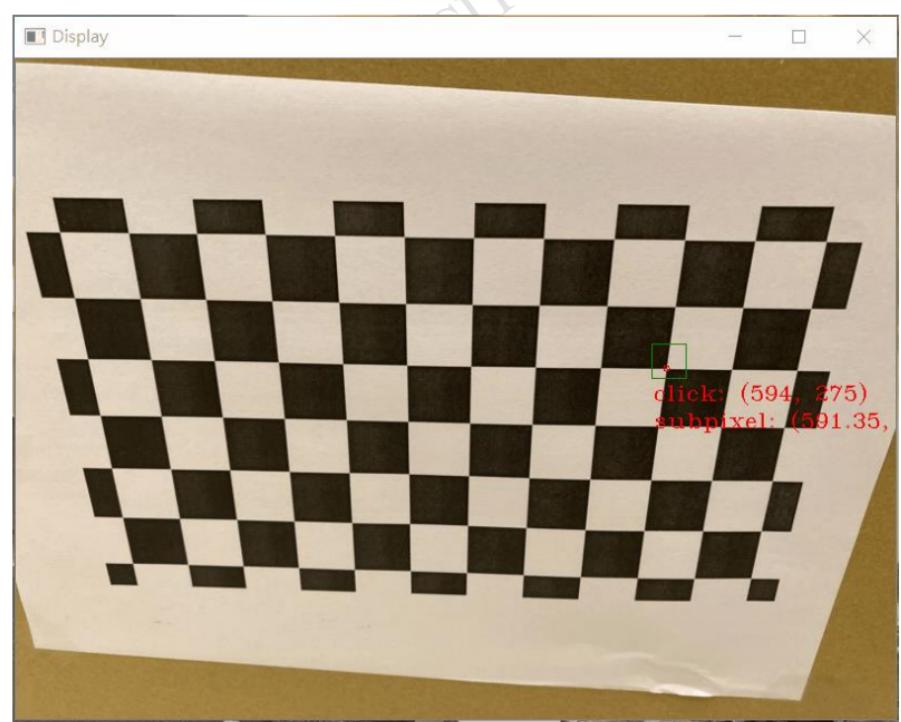
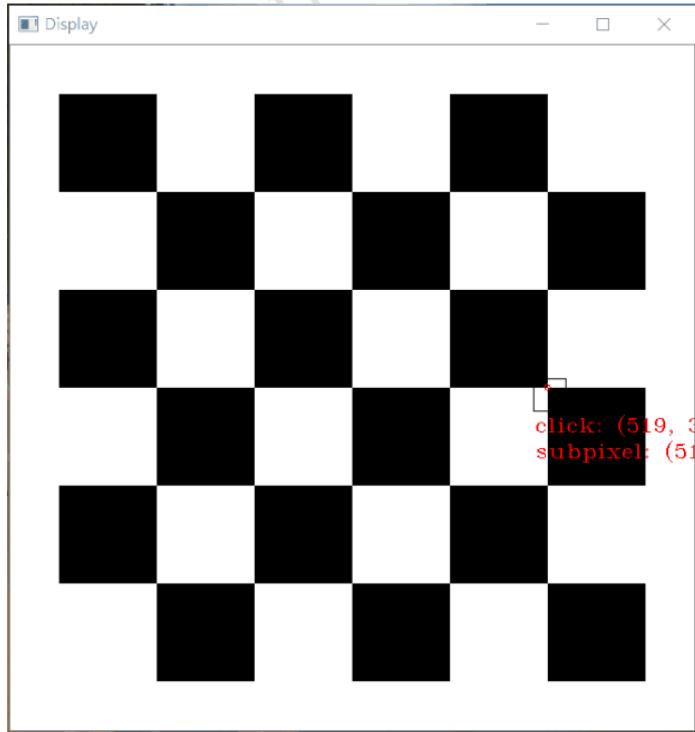
cvGoodFeaturesToTrack
+cvFindCornerSubPix



cvGoodFeaturesToTrack
(Harris criterion)
+cvFindCornerSubPix

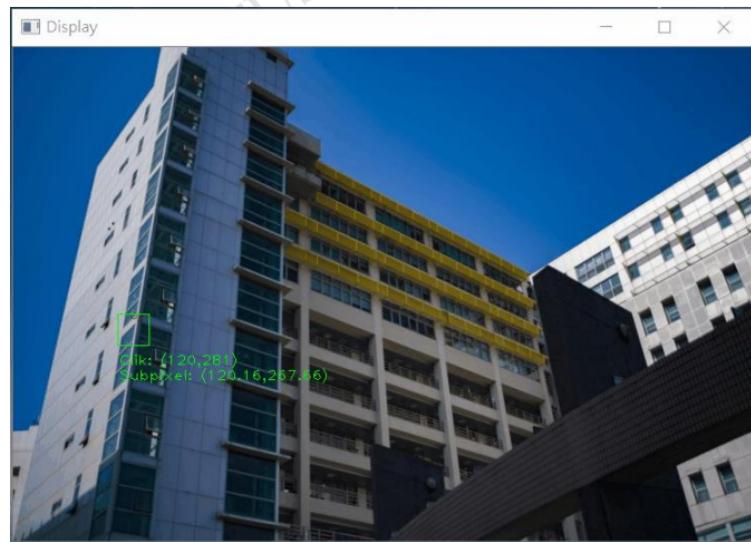
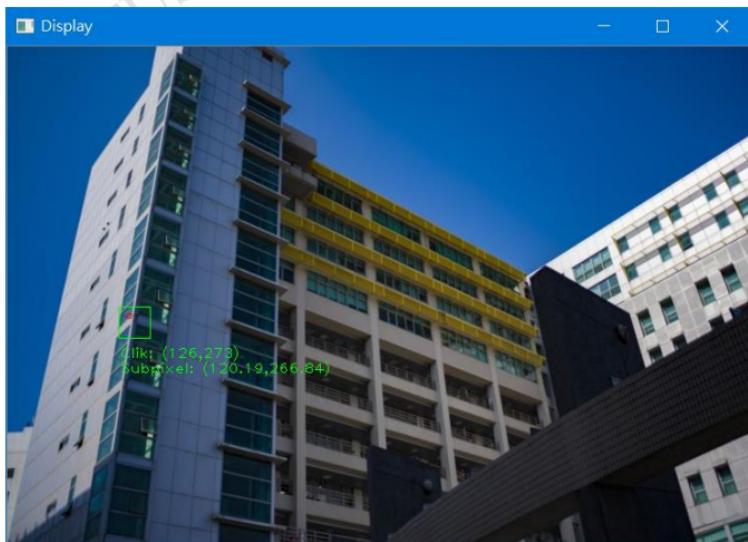


Subpixel in openCV





Subpixel in openCV





Feature detection

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



Hessian Matrix

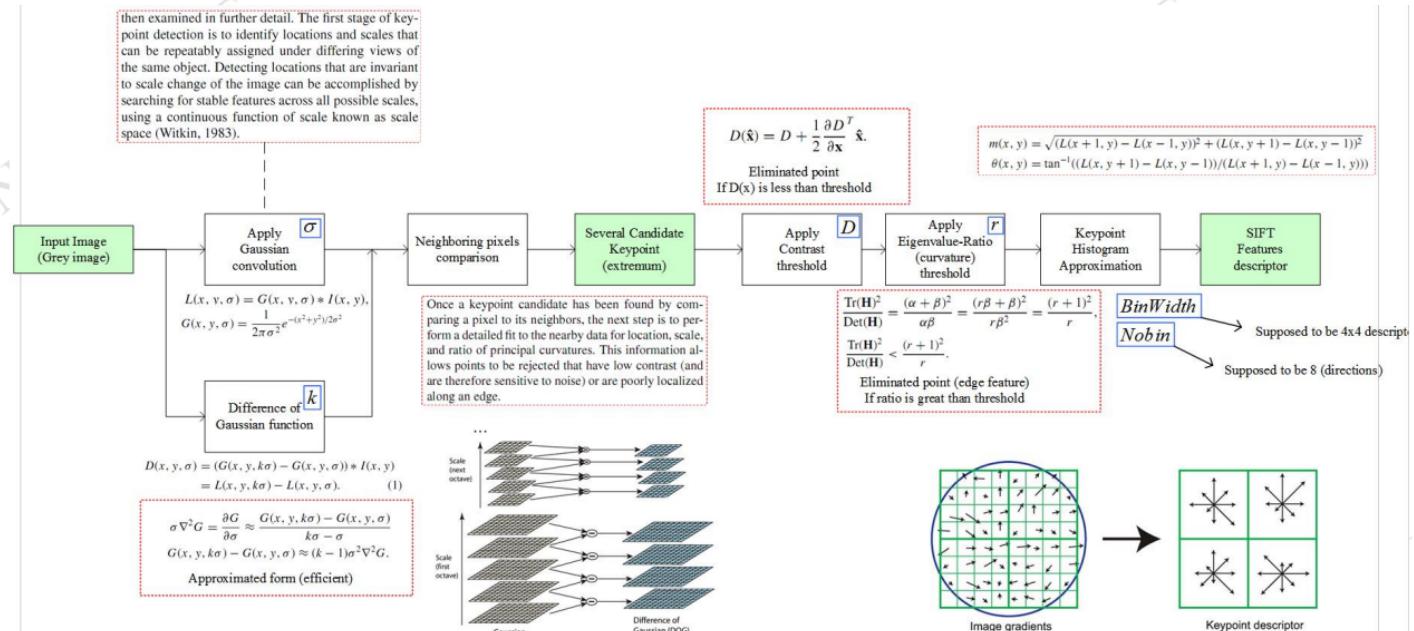
$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Reject when

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



Summary of SIFT



Optimized / Suggestion parameter

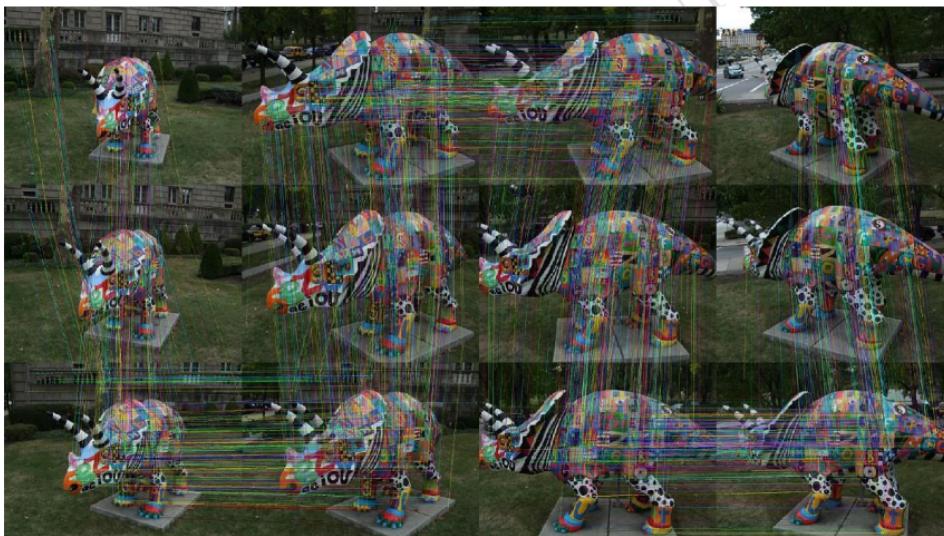
tave of scale space. The image doubling increases the number of stable keypoints by almost a factor of 4, but no significant further improvements were found with a larger expansion factor.

imgDBL



Feature detection

- SIFT: example





Feature detection

- SURF: Speeded Up Robust Features



//Sample code in openCV

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

Hessian Matrix

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



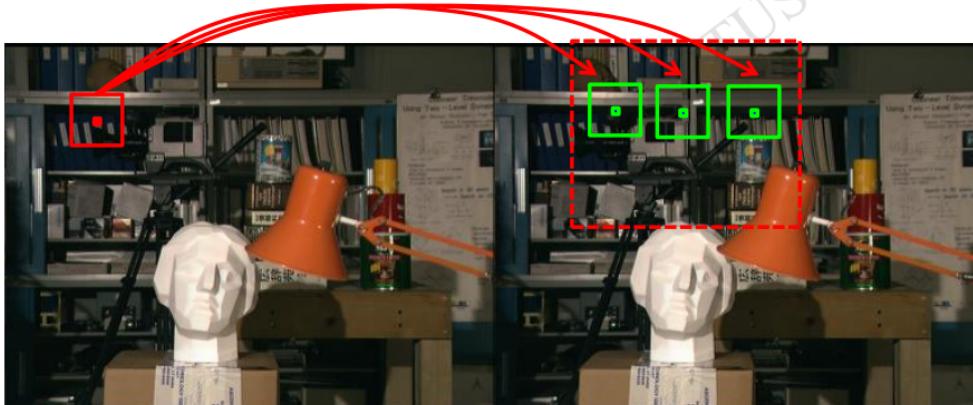
Feature matching

- 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)

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

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

$$\text{Disparity} = x_2 - x_1$$

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

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





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 :

$$SSD(\vec{v}_1, \vec{v}_2) = \sum_i (\vec{v}_1(i) - \vec{v}_2(i))^2 = \|\vec{v}_1 - \vec{v}_2\|^2 \leftarrow \begin{array}{l} \text{Squared} \\ \text{vector norm} \end{array}$$

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

- Normalized Cross Correlation is:

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

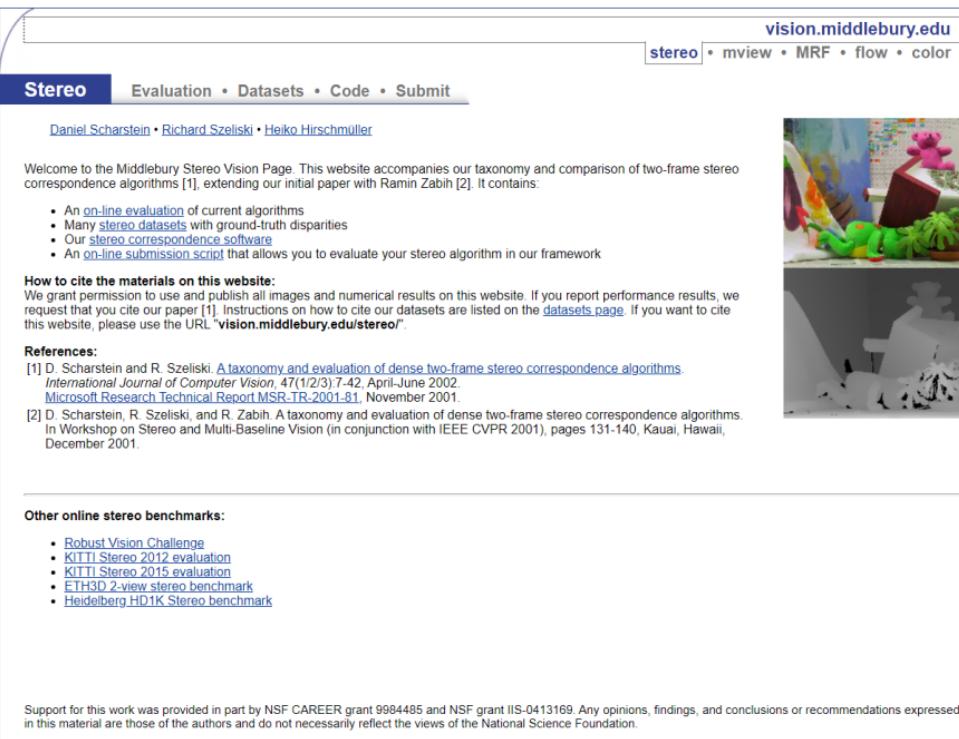


Feature matching

- 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.



Middlebury stereo benchmarks (stereo matching)



The screenshot shows the Middlebury Stereo Vision Page. At the top, there's a navigation bar with links for 'Stereo', 'Evaluation', 'Datasets', 'Code', and 'Submit'. Below the navigation, it says 'Daniel Scharstein • Richard Szeliski • Heiko Hirschmüller'. A welcome message from the authors is present. A bulleted list highlights features: an online evaluation of current algorithms, many stereo datasets with ground-truth disparities, our stereo correspondence software, and an online submission script. A section on how to cite the materials on this website follows, with instructions for reporting performance results and citing the paper. A 'References' section lists two papers by Scharstein et al. A 'References' section also lists other online stereo benchmarks like Robust Vision Challenge, KITTI Stereo 2012 evaluation, etc. At the bottom, a note about NSF grants is included.

Support for this work was provided in part by NSF CAREER grant 9984485 and NSF grant IIS-0413169. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

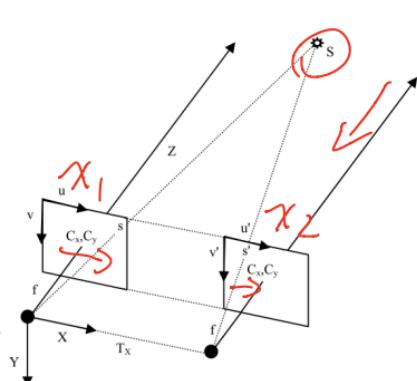
<https://vision.middlebury.edu/stereo/>



Stereoscopic image to 3D data

- Reprojecting of disparity into 3D space
 - Reprojection matrix in homogenous coordinate

$$|x_1 - x_2|$$



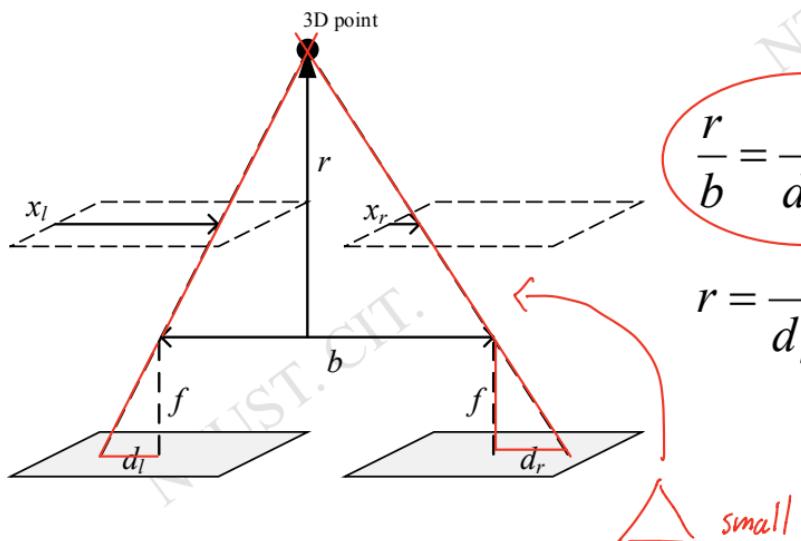
$$\mathbf{Q} \equiv \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & \frac{C_x - C'_x}{T_x} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \mathbf{Q} \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix}$$

The actual 3D point is $(X/W, Y/W, Z/W)$.



Stereoscopic image performance estimation

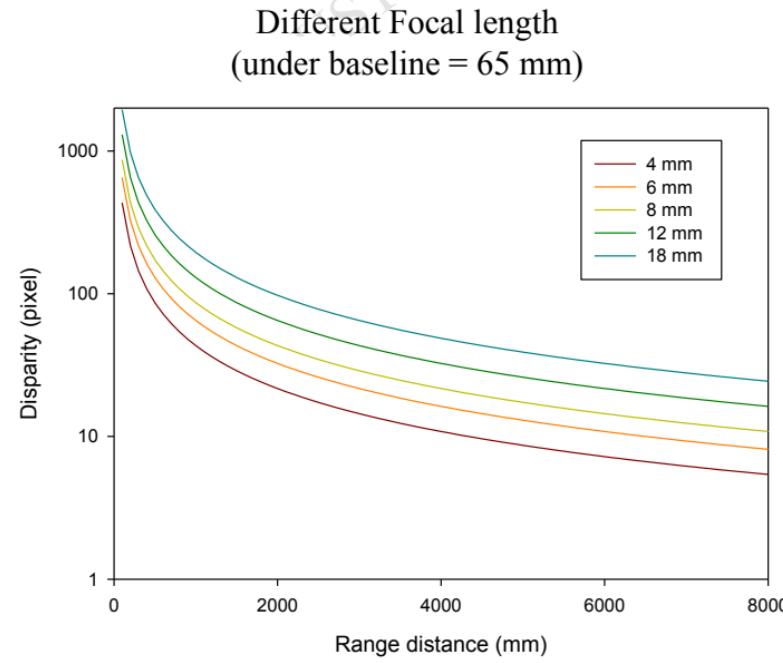
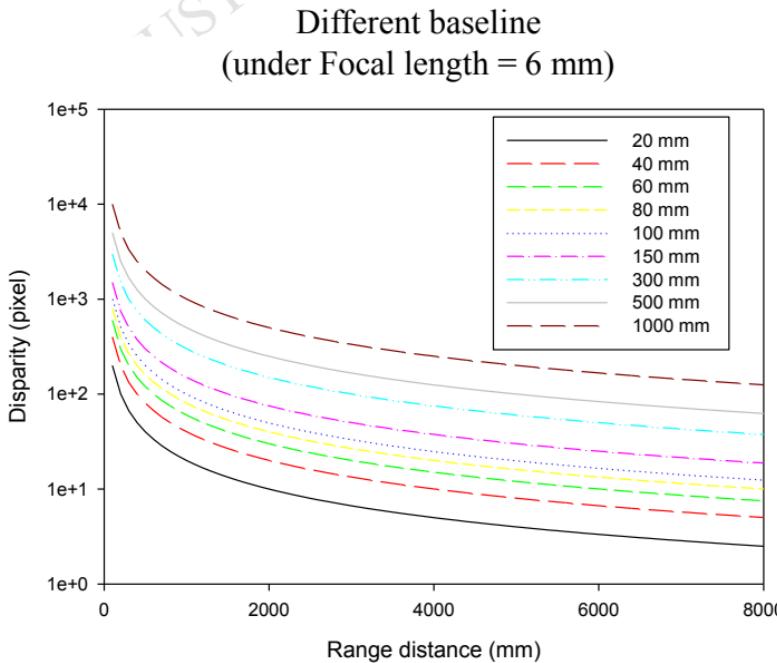


$$\frac{r}{b} = \frac{f}{d_l + d_r}$$

$$r = \frac{fb}{d_l + d_r} = \frac{fb}{x_l - x_r} = \frac{fb}{d}$$



Stereoscopic image performance estimation

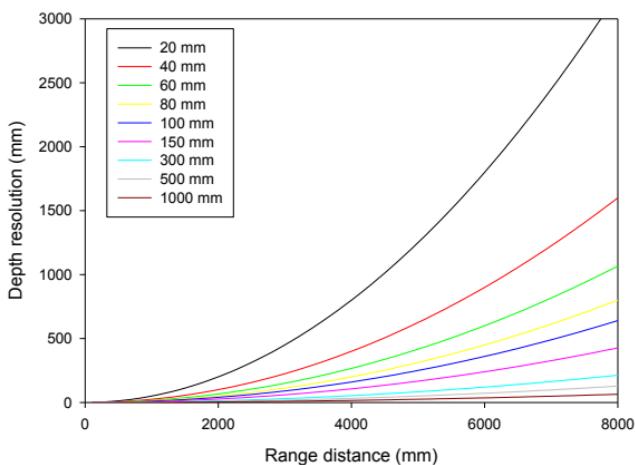


for pixel size = $6.0 \mu\text{m}$ CMOS



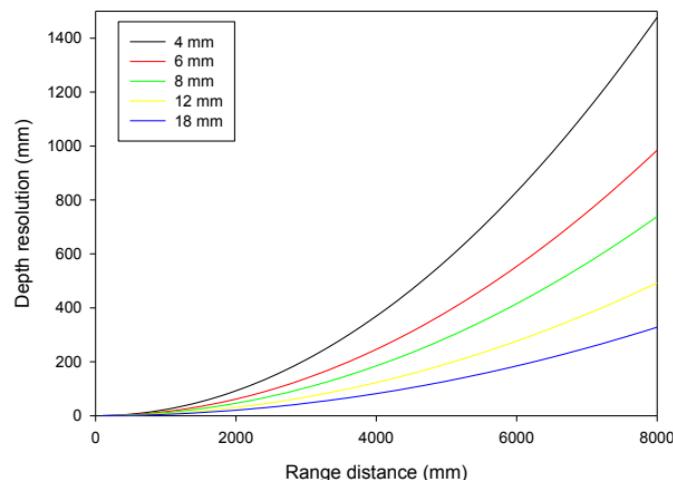
Stereoscopic image performance estimation

Different baseline
(under Focal length = 6 mm)



$$\Delta d = \frac{fb}{r + \Delta r} - \frac{fb}{r}$$
$$\frac{\Delta d}{fb} = \frac{1}{r + \Delta r} - \frac{1}{r} = \frac{-\Delta r}{(r + \Delta r)r}$$
$$\Delta r \approx -\frac{r^2}{fb} \Delta d$$

Different Focal length
(under baseline = 65 mm)

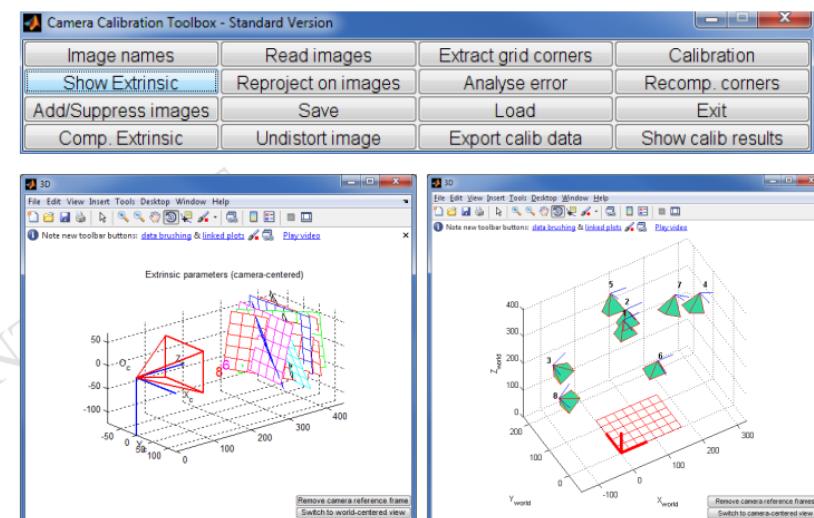


for pixel size = 6.0 μm CMOS



Camera calibration

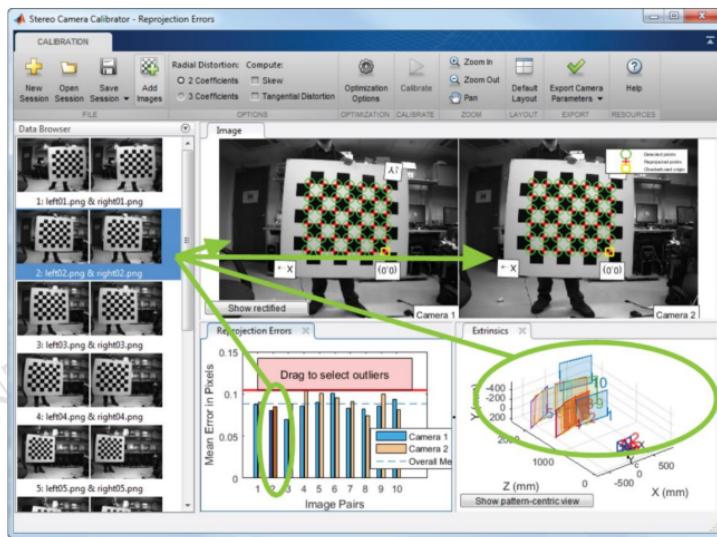
- Calibration toolbox for Matlab
 - To obtain intrinsic and extrinsic parameter
 - To obtain lens distortion compensation





Stereo camera calibration (by Matlab)

- Build-in toolbox
 - stereoCameraCalibrator





Feature matching

■ Block Matching in openCV

openCV function: `cvMatchTemplate`

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





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

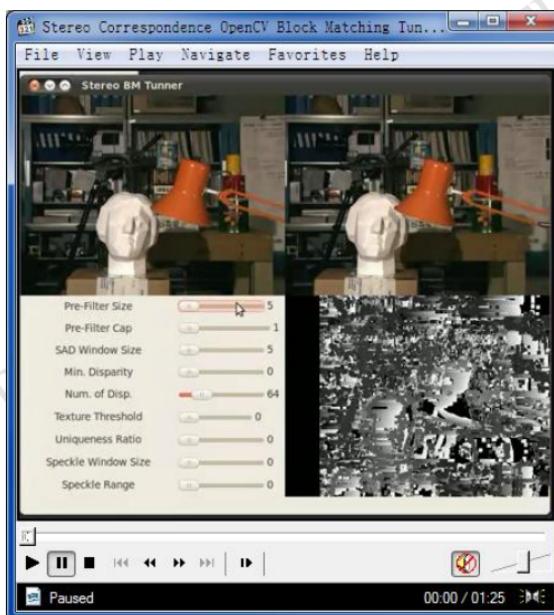
method=CV_TM_CCOEFF_NORMED

→ fast normalized correlation coefficient (FFT)



Feature matching

- Stereo-Block matching example



Video source from:

<http://www.youtube.com/watch?v=FX7AMktf24E>



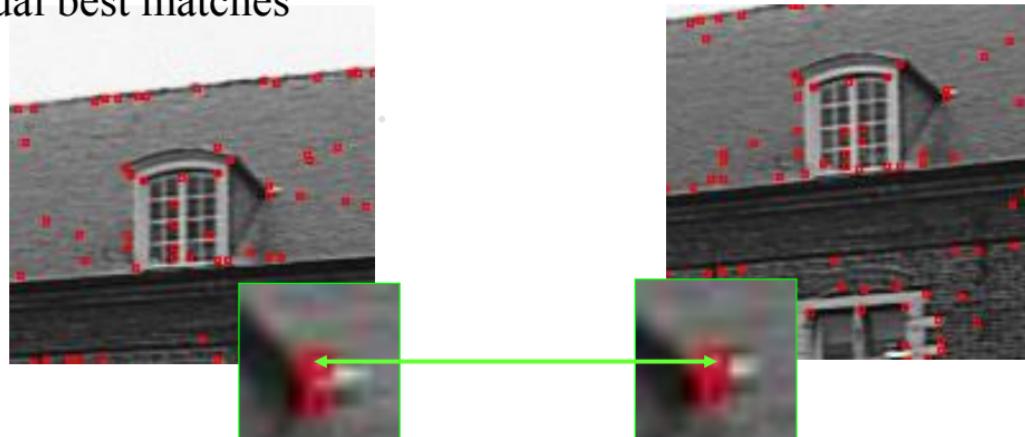
Feature matching

- 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

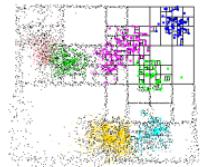
- Fast Searching Issue
 - ANN or FLANN (Fast Approximate Nearest Neighbor Search), KD-Tree



Harris + FLANN

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

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



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 arbitrary 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 supports a large 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/>

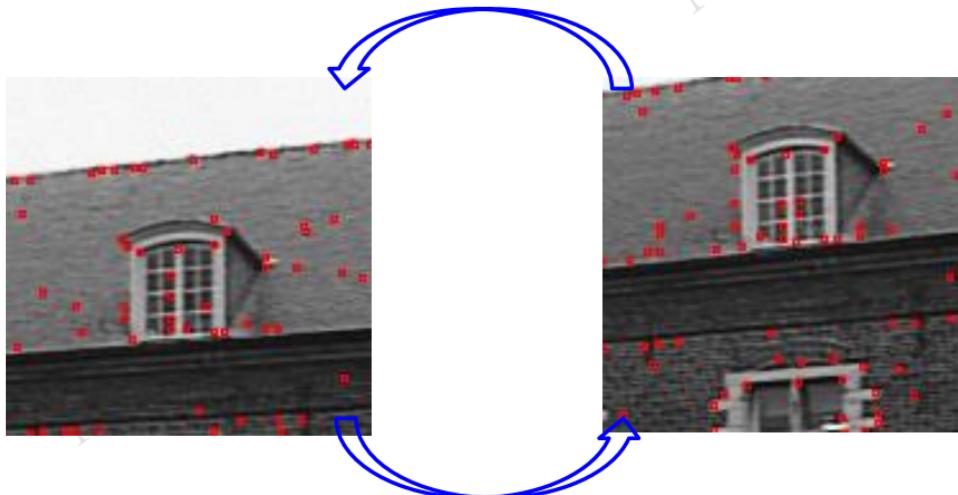
[1] K. Hajebi, Y. Abbasi-yadkori, H. Shahbazi, and H. Zhang, “Fast approximate nearest-neighbor search with k-nearest neighbor graph,” in *International Joint Conference on Artificial Intelligence*, 2009, no. C, pp. 1312-1317.

[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

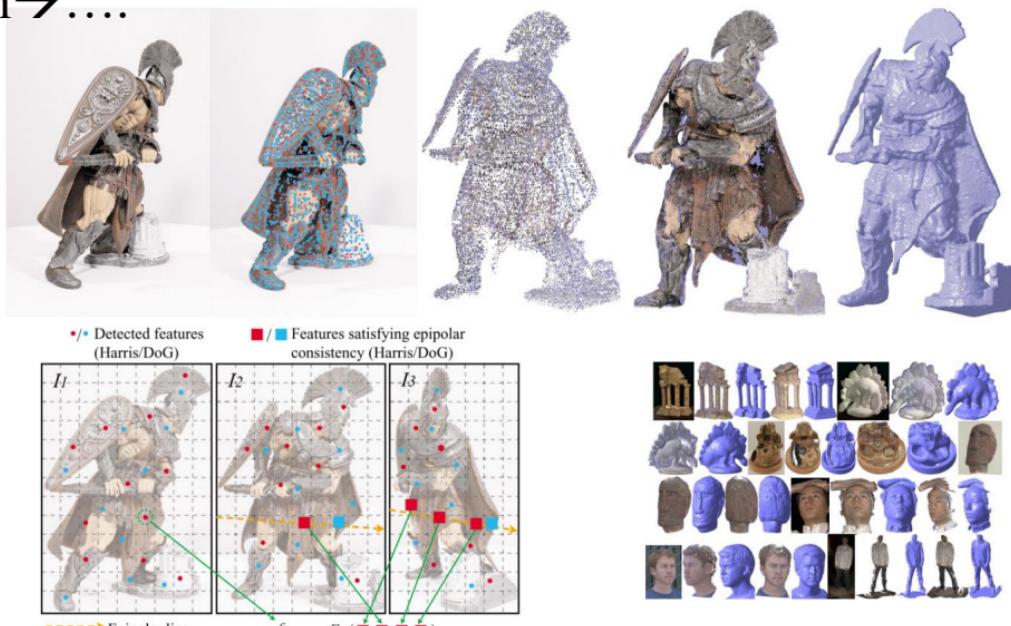


(f) RANSAC inliers 2



Feature matching

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





Feature tracking

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



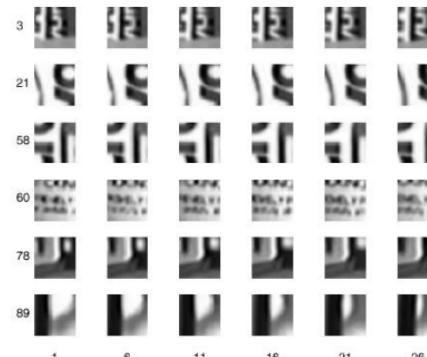
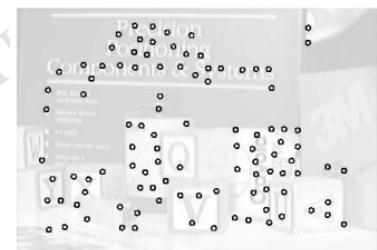
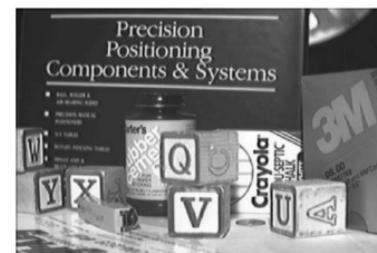
Feature tracking

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



Feature tracking

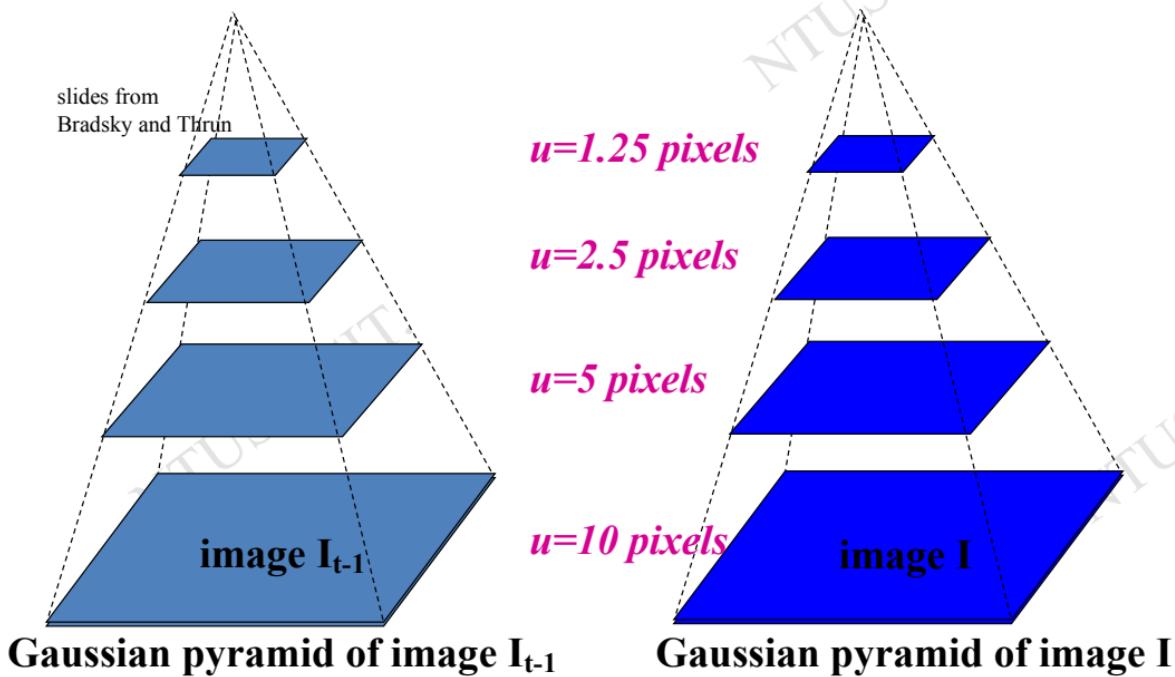
- Good features to keep tracking
 - Perform affine alignment between first and last frame. Stop tracking features with too large errors





Feature tracking

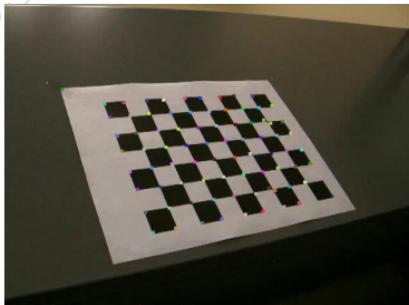
- Coarse-to-fine optical flow estimation



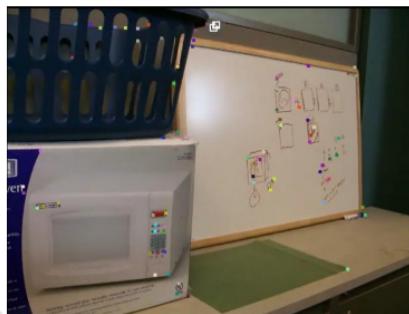


Feature tracking

■ KLT tracking example



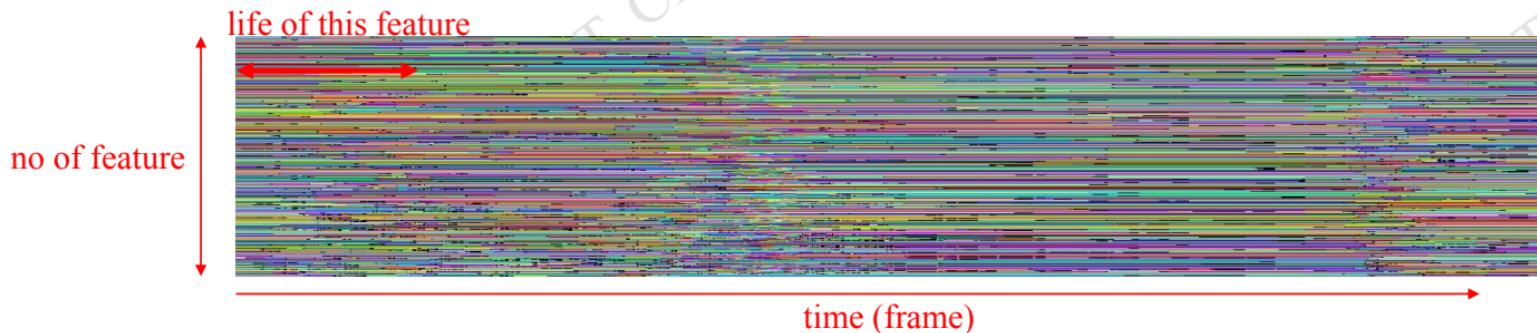
just enough



sparse feature



dense feature



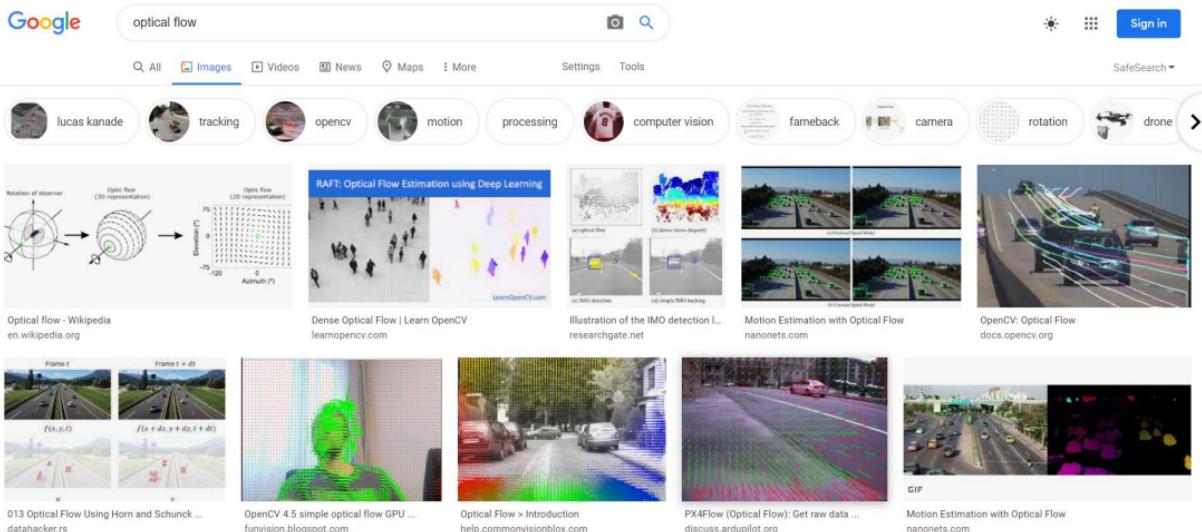


Feature tracking

kLT
ppp



- Optical flow (a prediction tracking model)



Optical flow function in openCV

`cvCalcOpticalFlowBM`
→using block matching

`cvCalcOpticalFlowHS`
→using Horn and Schunck

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



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

