SPATIAL SENSING

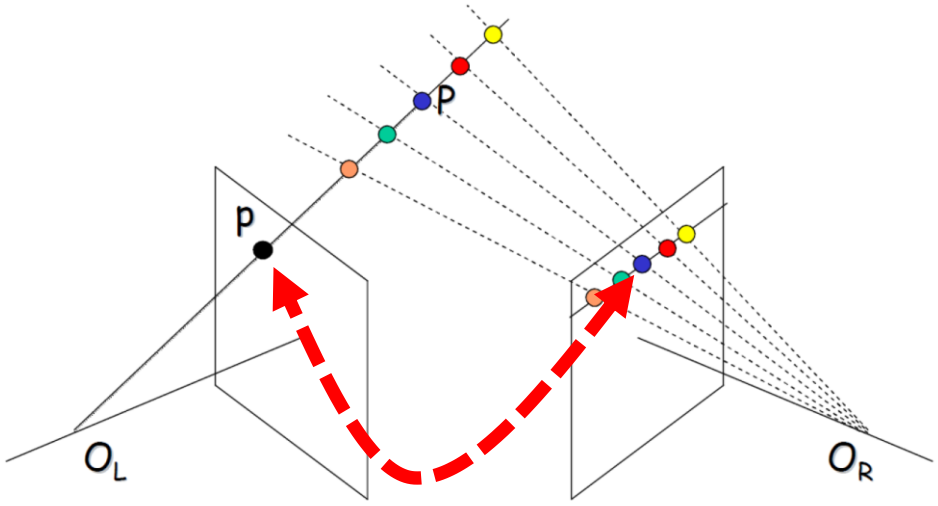
3D SENSOR DATA REPRESENTATION AND MODELLING

Dr TIAN Jing

tianjing@nus.edu.sg

Q1: Distance estimation (between physical point to camera)

- Ambiguity: Any point on the ray (the line from the camera centre O_L to the point P) can be projected on the same the image point.
- Solution: A second camera can resolve the ambiguity, enabling measurement of depth via triangulation.

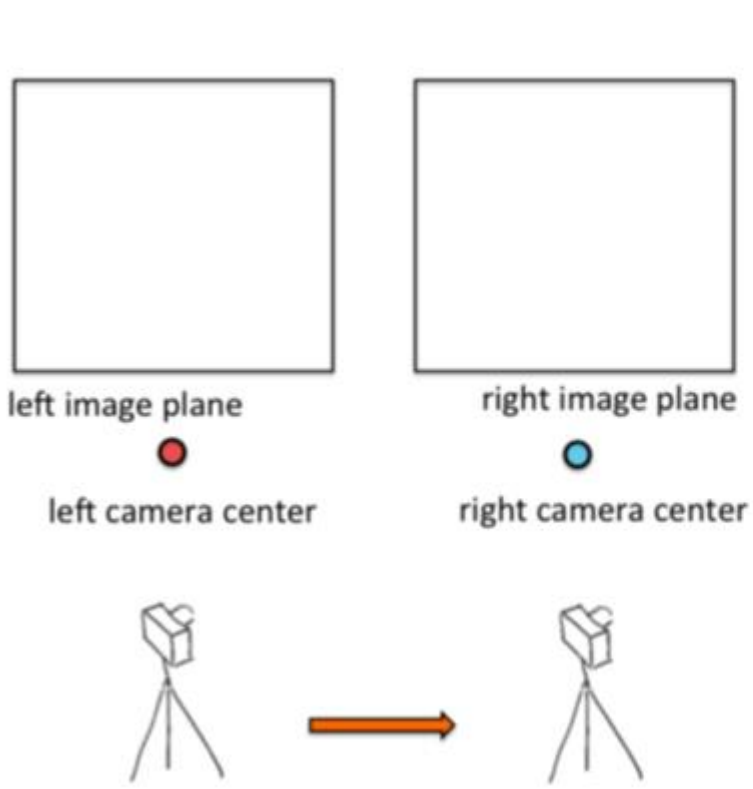


Reference: http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf

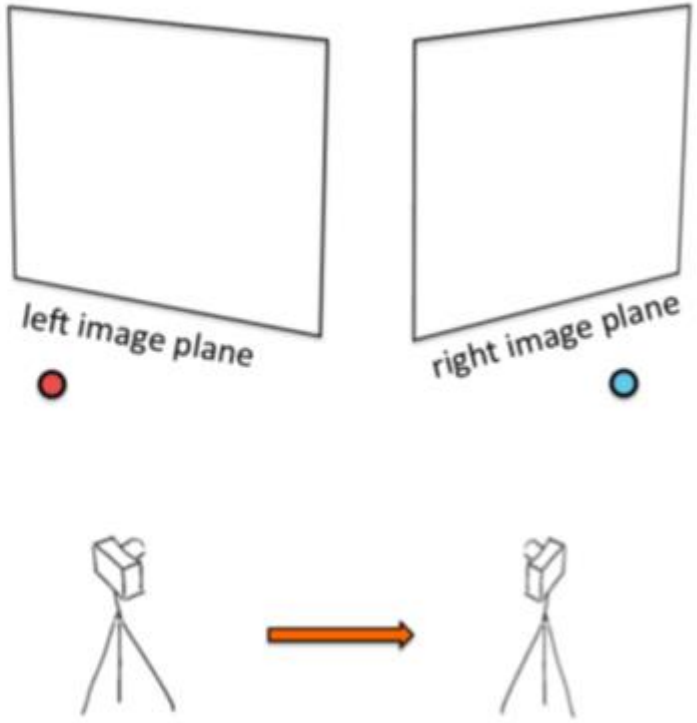
Stereo vision: Camera system

Two popular stereo camera systems

Parallel stereo camera system

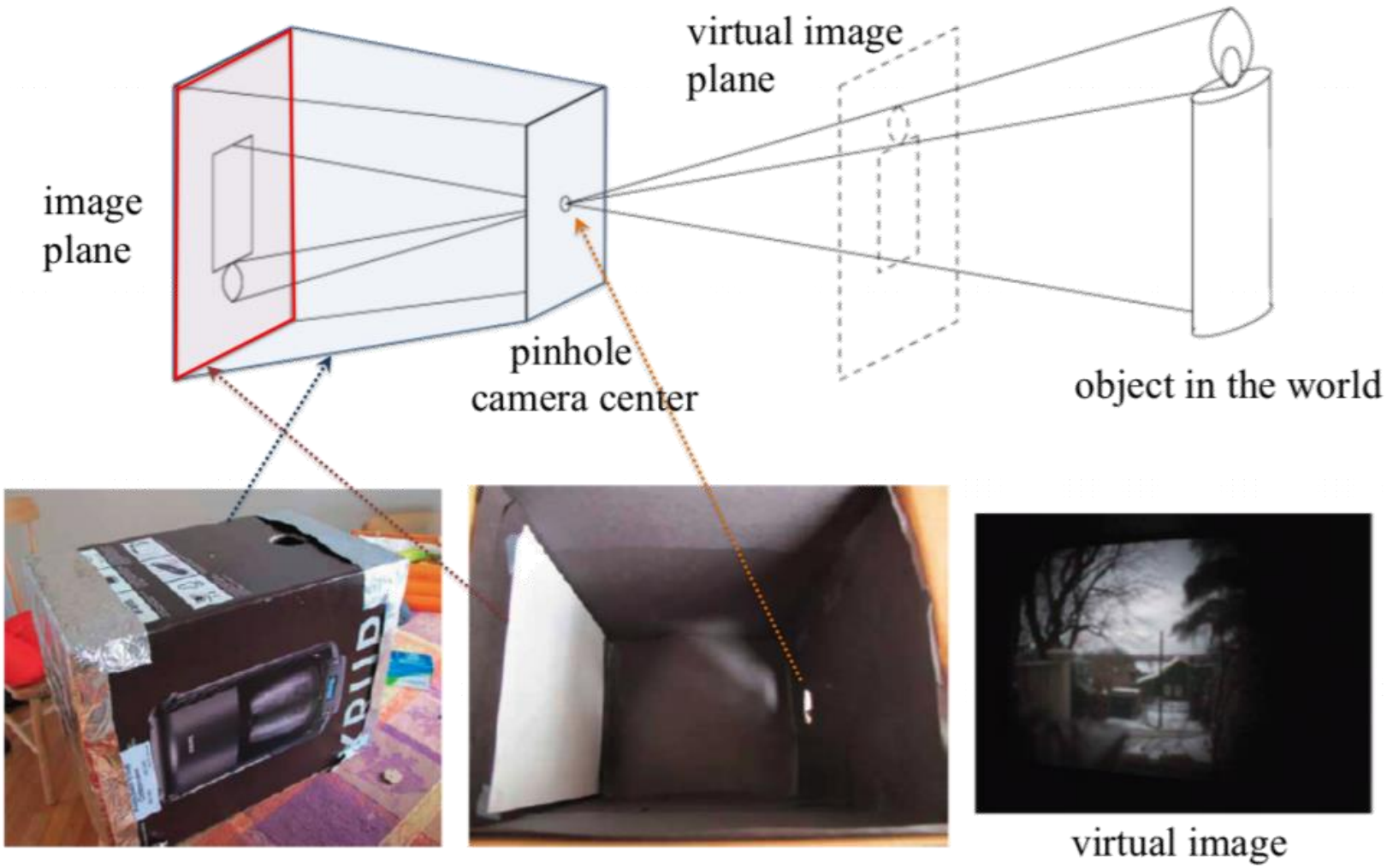


General stereo camera system



Reference: http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf

Pinhole camera model

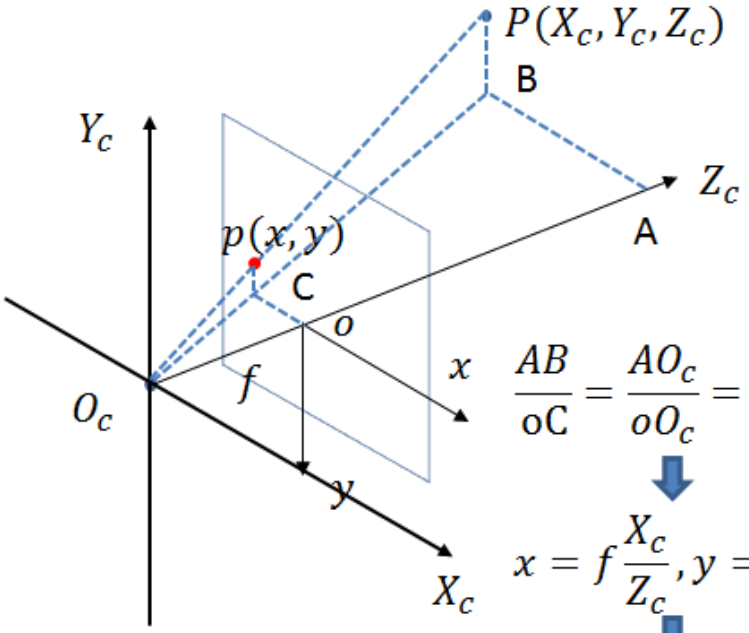


Pinhole camera model

Pinhole camera model: Convert from camera reference system $P(X_c, Y_c, Z_c)$ to the image reference system $P(x, y)$. It depends on camera model focal length f.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f/z_c & 0 & 0 & 0 \\ 0 & f/z_c & 0 & 0 \\ 0 & 0 & 1/z_c & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

homogeneous coordinates (see next slide)



$$\frac{AB}{oC} = \frac{AO_c}{oO_c} = \frac{PB}{pC} = \frac{X_c}{x} = \frac{Z_c}{f} = \frac{Y_c}{y}$$

Triangle similarity theorem

$$x = f\frac{X_c}{Z_c}, y = f\frac{Y_c}{Z_c}$$

Re-arrange as matrix format

$$Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

$\Delta ABO_c \sim \Delta oCO_c$

$\Delta PBO_c \sim \Delta pCO_c$

Reference: Module 2, Vision Algorithms for Mobile Robotics, http://rpg.ifi.uzh.ch/teaching.html

Intrinsic matrix

Suppose for the CMOS/CCD sensor, each pixel has a physical size d_x, d_y, the image plane origin is located at the position $(u_0, v_0, 1)$, then $u = \frac{x}{d_x} + u_0, v = \frac{y}{d_y} + v_0$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_0 \\ 0 & 1/d_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous coordinates (see next slide)

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_0 \\ 0 & 1/d_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f/z_c & 0 & 0 & 0 \\ 0 & f/z_c & 0 & 0 \\ 0 & 0 & 1/z_c & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f/d_x & 0 & u_0 & 0 \\ 0 & f/d_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- α_x, α_y focal length in pixels
- γ skew between x and y axes (often zero)
- u_0, v_0 principal point (typically center of image)

Example: Given an image resolution of 640×480 pixels and a focal length of 210 pixels, the intrinsic matrix could be

$$K = \begin{bmatrix} 210 & 0 & 320 & 0 \\ 0 & 210 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Reference: https://www.mathworks.com/help/vision/ug/camera-calibration.html

Appendix: Homogeneous coordinate

Homogeneous coordinate

- Stick '1' at the end of the original coordinate vector.
- Advantage: Express both translation and rotation using the matrix multiplication.

Translation only

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \mathbf{P}' = \begin{bmatrix} t_x + x \\ t_y + y \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{T} + \mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \mathbf{P}' = \begin{bmatrix} t_x + x \\ t_y + y \\ 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{T} \cdot \mathbf{p}$$

$$\begin{bmatrix} t_x + x \\ t_y + y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation only: around z axis

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \mathbf{P}' = \begin{bmatrix} \cos\theta\, x - \sin\theta\, y \\ \cos\theta\, y + \sin\theta\, x \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{R} \cdot \mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \mathbf{P}' = \begin{bmatrix} \cos\theta\, x - \sin\theta\, y \\ \cos\theta\, y + \sin\theta\, x \\ 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{R} \cdot \mathbf{p}$$

$$\begin{bmatrix} \cos\theta\, x - \sin\theta\, y \\ \cos\theta\, y + \sin\theta\, x \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Stereo vision: Camera system

Given two calibrated parallel cameras, i.e. the right camera is some distance to the right of the left camera. According to triangle similarity theorem between blue triangle and red triangle, we have

$$\frac{T}{Z_c} = \frac{T + x_r^* - x_l^*}{Z_c - f}$$

$$x_l = \frac{x_l^*}{d_x},\ x_r = \frac{x_r^*}{d_x}$$

$$Z_c = \frac{fT}{d_x(x_l - x_r)}$$

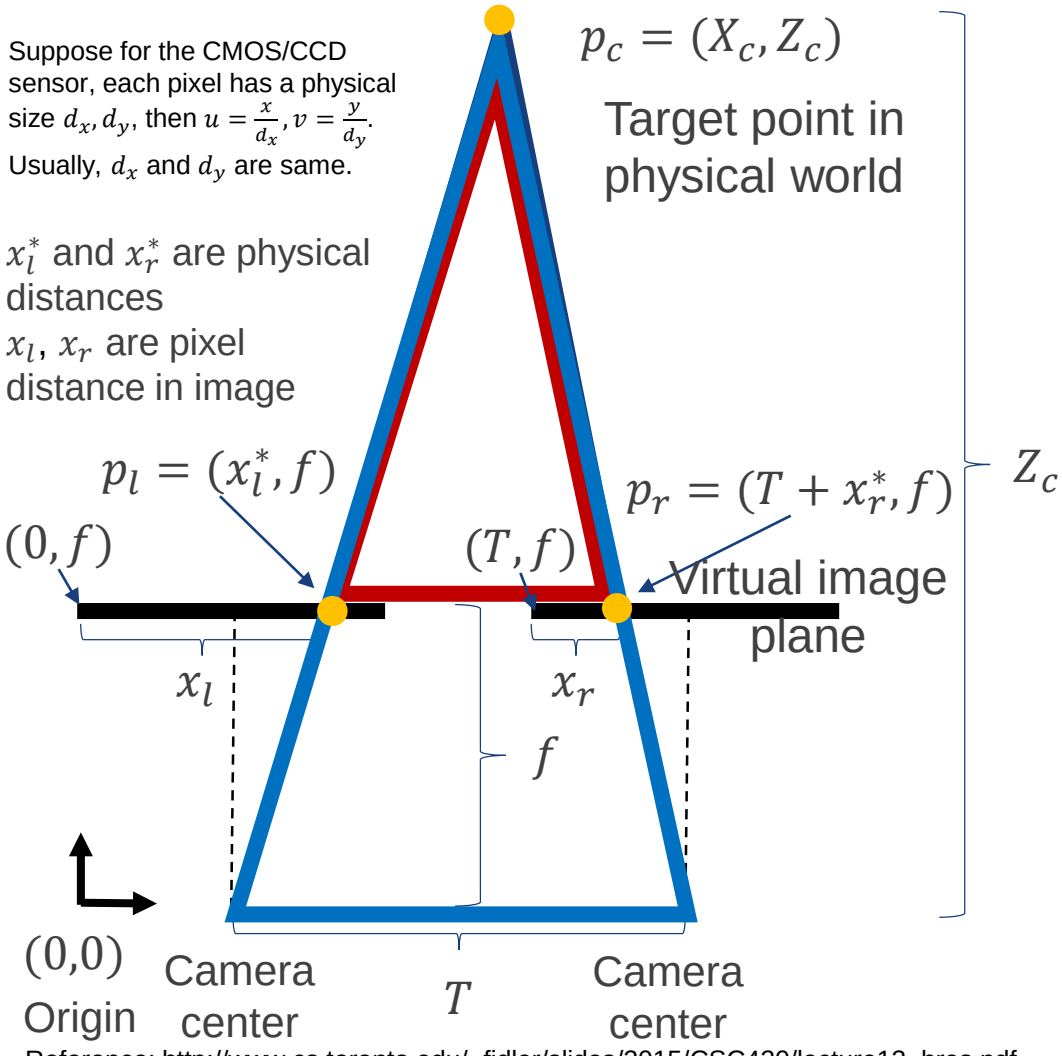
	Descriptions	Unit
Z_c	Distance between point p to camera	Physical distance, meter
T	Baseline distance between two cameras	Physical distance, meter
f	Focal length of the camera	Physical distance, meter
x_l, x_r	Locations of point p_l, p_r in images	Pixels
d_x, d_y	Physical size of a pixel in camera sensor CMOS/CCD	Physical distance per pixel

An illustration from bird-view (top-view)

Suppose for the CMOS/CCD sensor, each pixel has a physical size d_x, d_y, then $u = \frac{x}{d_x}, v = \frac{y}{d_y}$. Usually, d_x and d_y are same.

x_l^* and x_r^* are physical distances

x_l, x_r are pixel distance in image



Reference: http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf

Disparity estimation

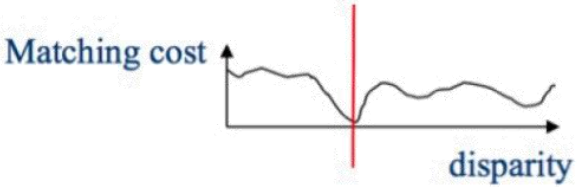
For each point $\mathbf{p}_l = (x_l, y_l)$, how to get $\mathbf{p}_r = (x_r, y_r)$ by matching?

- Idea: Image patch centered at (x_r, y_r) should be similar to the image patch centered at (x_l, y_l). We scan along the horizontal line and compare patches to the one in the left image and looking for a most similar patch.
- The matching cost can be defined as *SSD* (sum of squared differences), e.g.,

$$SSD(\text{patch}_l, \text{patch}_r) = \sum_x \sum_y \left(I_{\text{patch}_l}(x, y) - I_{\text{patch}_r}(x, y) \right)^2$$

left image

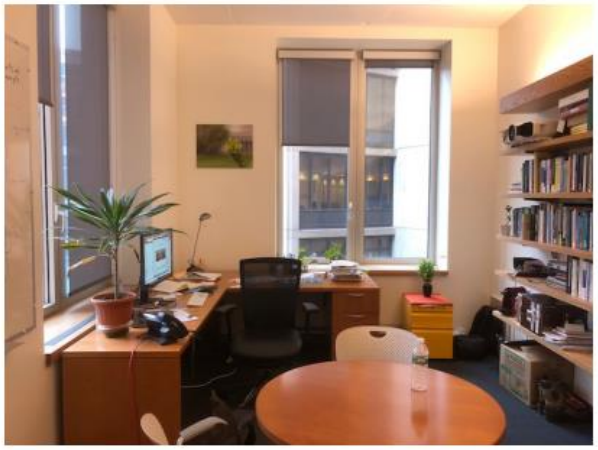
right image



Reference: http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf

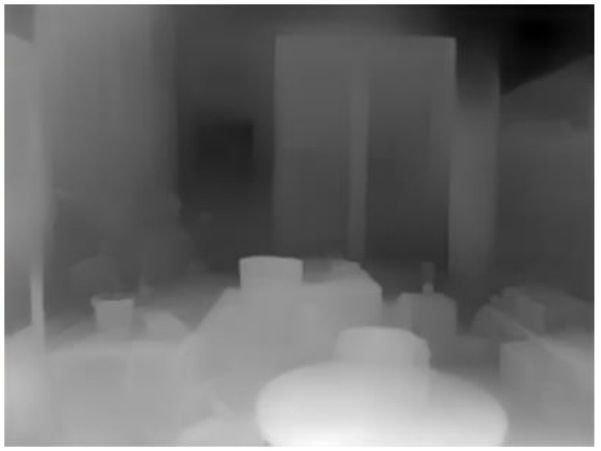
Example: Disparity estimation from stereo images

Left image



Right image

Disparity map

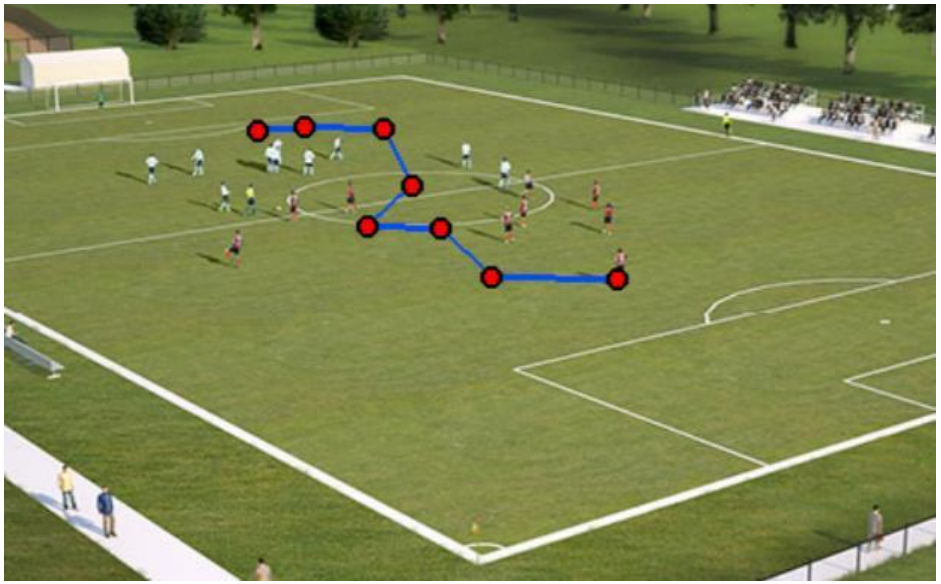


Disparity value at each pixel location

Reference: Imaging geometry, 6.869/6.819 Advances in Computer Vision, http://6.869.csail.mit.edu/sp22/lectures/L3/imaging_geometry2022.pdf

Q2: Distance estimation (between two pixel positions in the image)

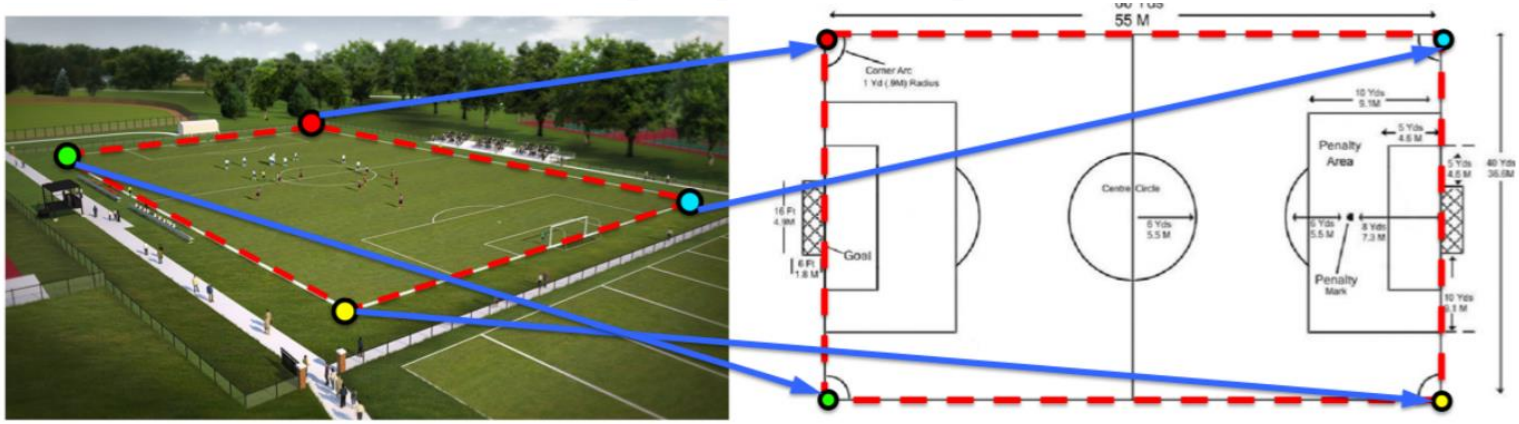
What is social distance (meters) between two detected persons?



How long (meters) has the football player run?

Reference: http://www.cs.toronto.edu/~fidler/slides/2021Winter/CSC420/lecture9.pdf

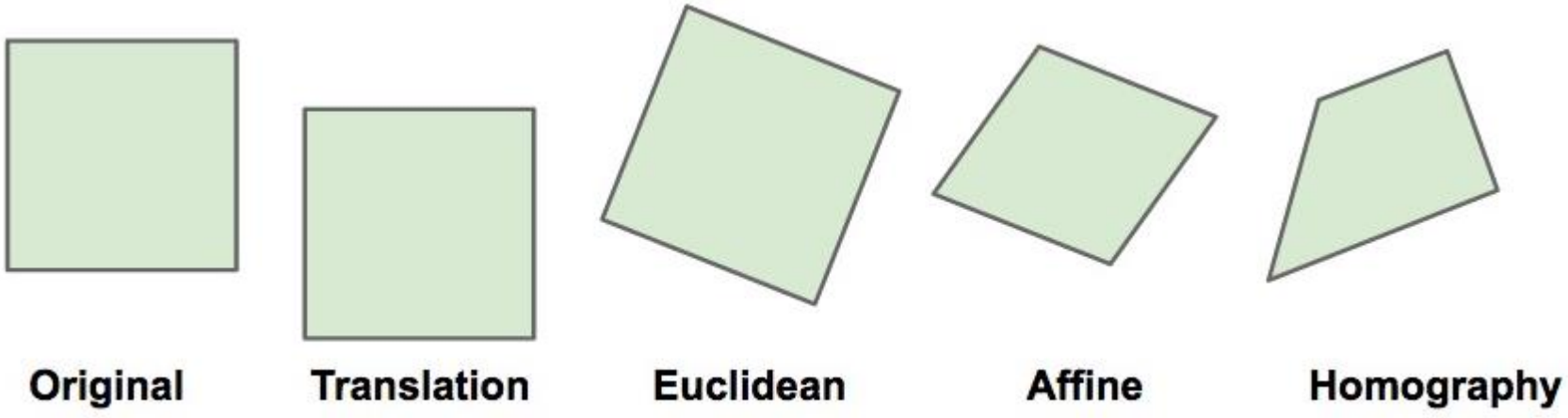
Intuition: Transform the CCTV view to a bird-view (top-view)



Reference:

- http://www.cs.toronto.edu/~fidler/slides/2021Winter/CSC420/lecture9.pdf
- https://github.com/Mentos05/SAS_DeepLearning/tree/master/Social%20Distancing%20Demo

Various transformations



Transformation	Preserves
Translation	Orientation
Euclidean (rotation, translation)	Lengths
Affine (rotation, translation, scaling)	Parallelism
Homography (projective)	Straight lines

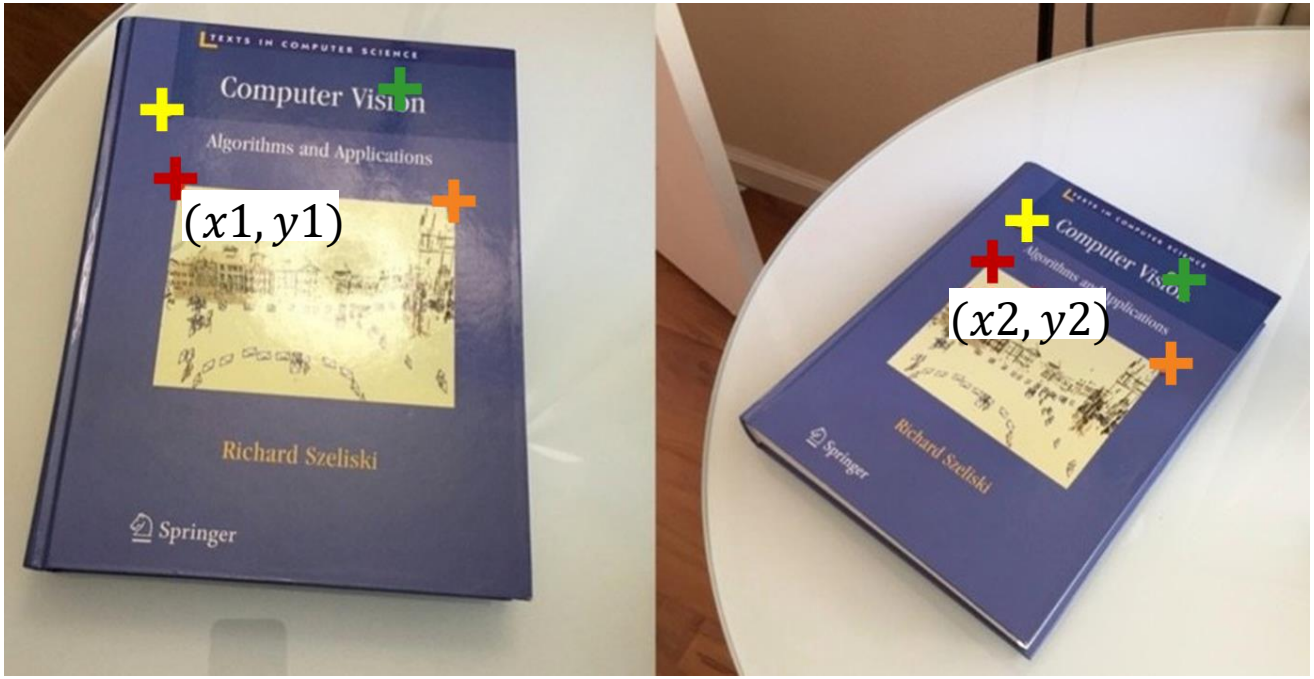
Reference: https://scikit-image.org/docs/stable/auto_examples/transform/plot_transform_types.html

Homography matrix

cv2.findHomography()

- A Homography matrix is a transformation matrix (3×3) maps the point located at $(x1, y1)$ in one image to the corresponding point located at $(x2, y2)$ in the other image.
- It is true for ALL sets of corresponding points as long as they lie on the same plane in the real world.

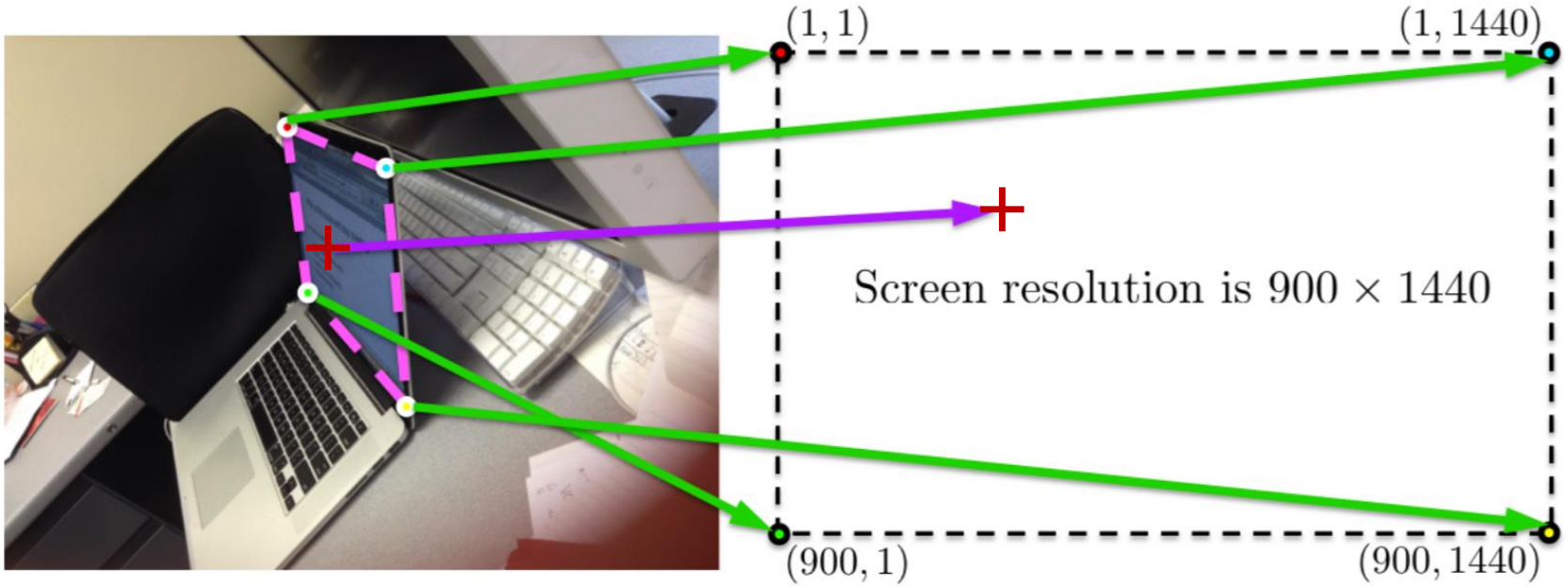
$$\begin{pmatrix} x1 \\ y1 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} x2 \\ y2 \\ 1 \end{pmatrix}$$



Reference:
https://learnopencv.com/homography-examples-using-opencv-python-c/

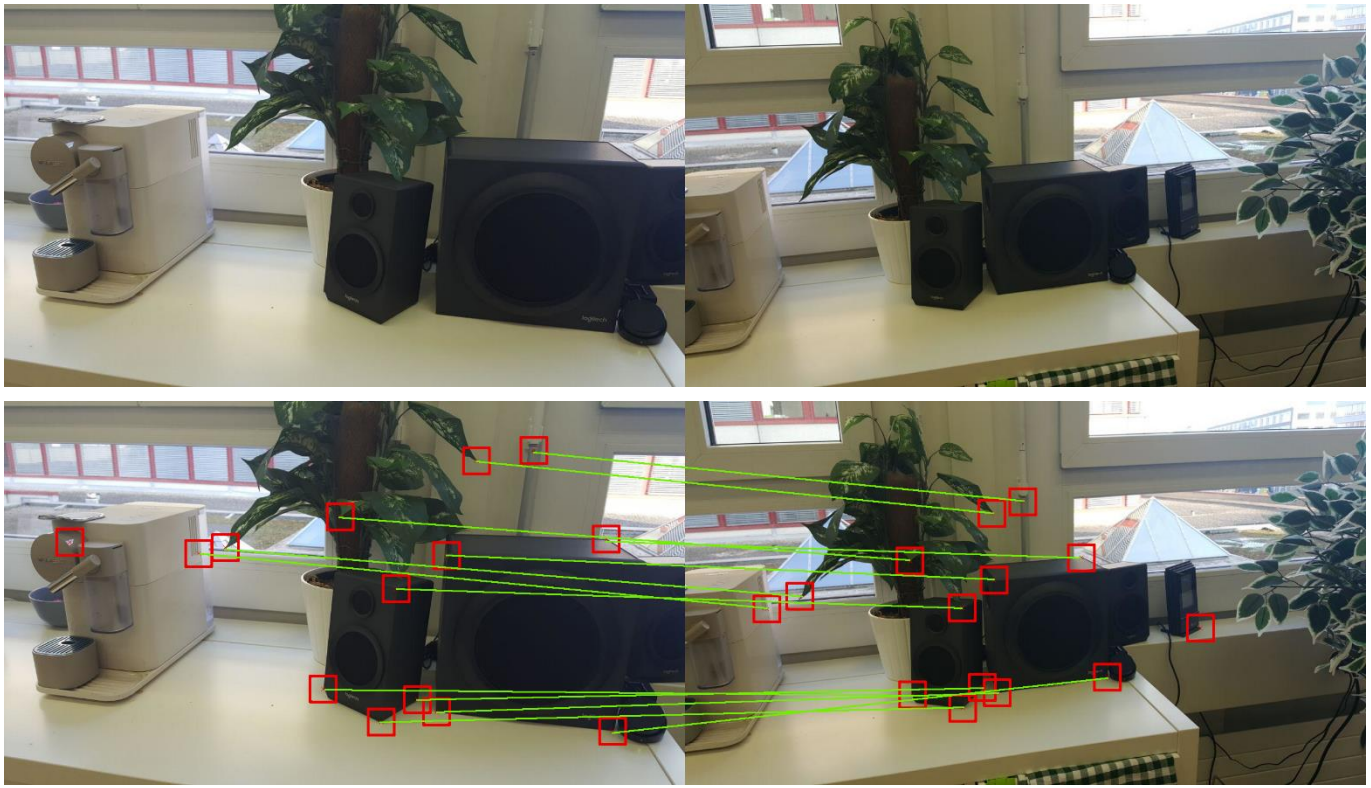
Homography: Perspective correction

- Collect four corners of the object plane (e.g., laptop screen), called *pts_src*.
- Set the target plane (resolution and aspect ratio is decided by users based on domain knowledge), called *pts_dst*.
- Obtain the homography matrix using *pts_src* and *pts_dst* via OpenCV function *cv2.findHomography()* .
- Map the position (red cross '+') from the source image to the target plane to obtain its location.



Reference: http://www.cs.toronto.edu/~fidler/slides/2021Winter/CSC420/lecture9.pdf

Fundamental task: Keypoint matching in multiple-view images



Input data
(multiple-view images)

Matched pairs

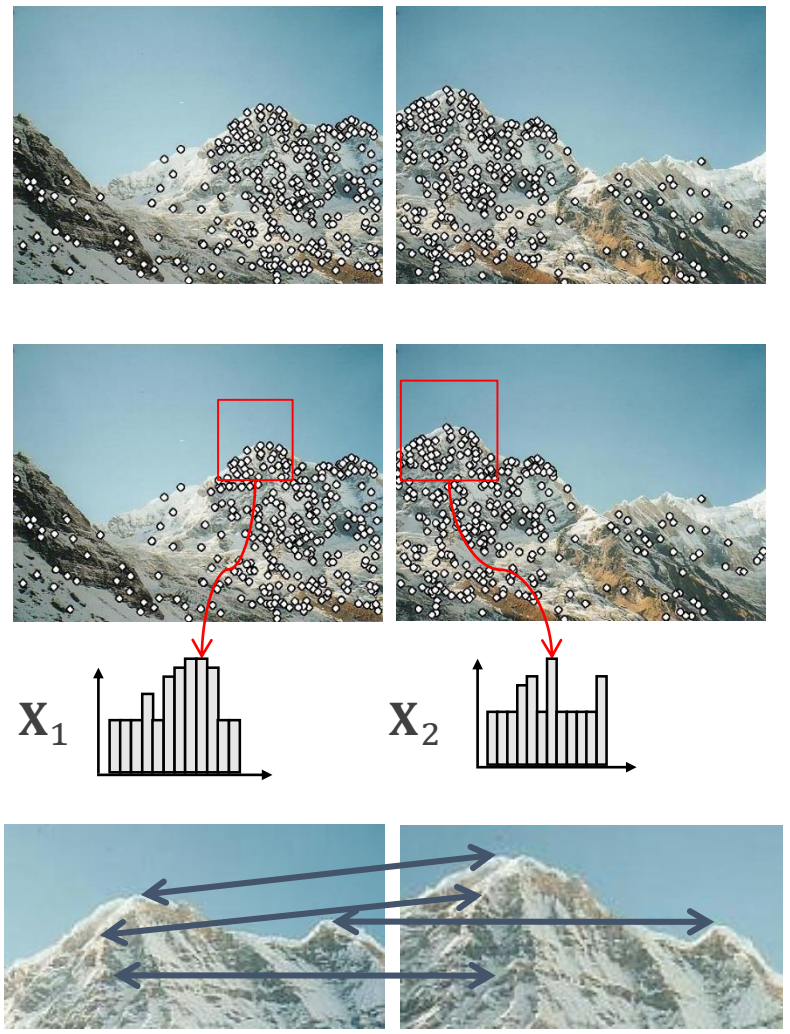
With matched pairs of keypoints in multiple-view images, we are able to

- Given a pair of matched keypoints (generated from the same physical point) in multiple-view images, we can estimate their disparity, then further estimate the distance between the physical point to the camera.
- Estimate the transformation (e.g., Homography matrix) between multiple-view images, which can help to estimate distance between two pixel positions in the image.

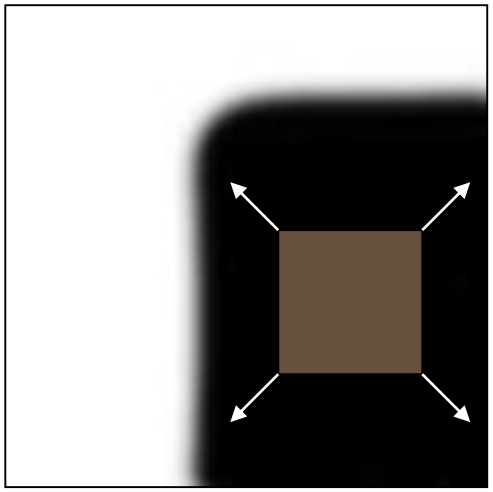
Image: http://rpg.ifi.uzh.ch/docs/teaching/2019/01_introduction.pdf

Identify matched keypoints via feature matching

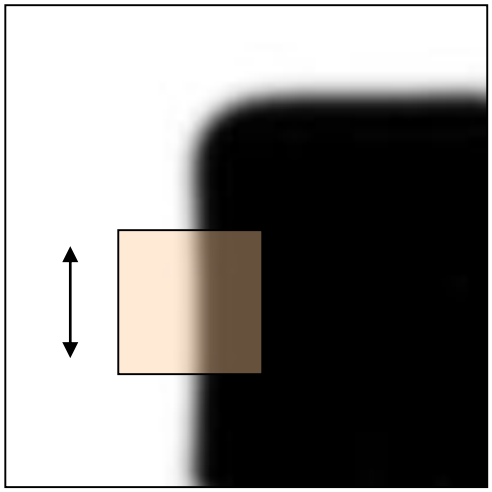
1) Detection:
   Find a set of distinctive keypoints from two images independently.
2) Description:
   Extract feature descriptor around each keypoint as vector, such as $\mathbf{X}_1, \mathbf{X}_2$.
3) Matching:
   Compute distance between feature vectors to find correspondence based on user-defined threshold T, i.e., $d(\mathbf{X}_1, \mathbf{X}_2) < T$.



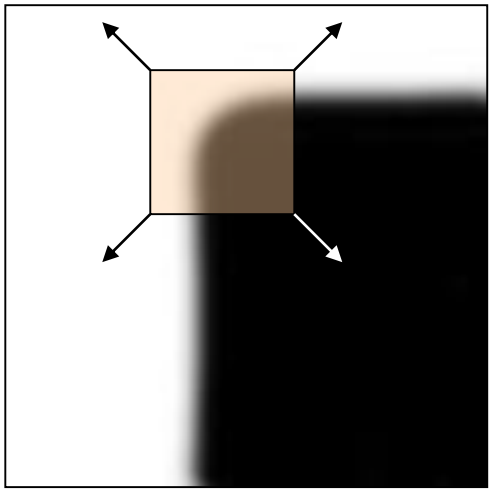
Keypoint detection: Harris corner detector



Flat: No change in all directions



Edge: No change along the edge direction



Corner: Significant change in all directions

Objective: Find patches (a window $\Omega(x, y)$ centered at the location (x, y)) that generate a large variation when it is moved around with a shift value (u, v).

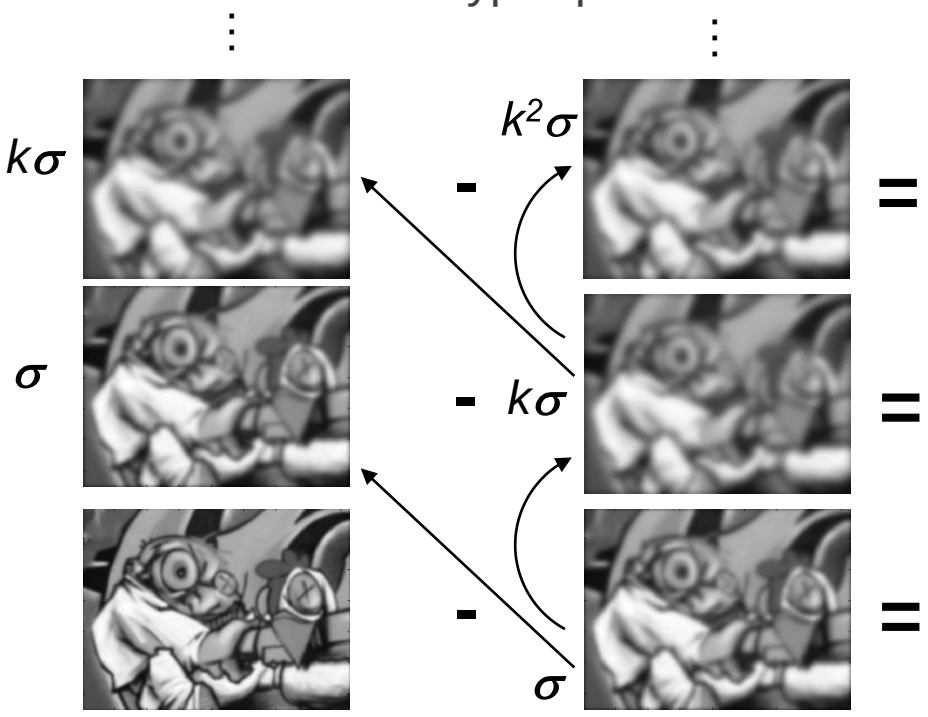
$$E_{u,v} = \sum_{\Omega(x,y)} w(x, y)(I(x + u, y + v) - I(x, y))^2$$

- $E_{u,v}$ is the difference between the original patch centered at $I(x, y)$ and that covered by the shifted window centered at $I(x + u, y + v)$.
- (u, v) are the window's displacements in the x, y directions, respectively.
- $w(x, y)$ is the mask function at position (x, y), e.g., uniform function or Gaussian function.
- We look for a patch with a large variation among $E_{0,0}$, $E_{1,0}$, $E_{0,1}$, $E_{1,1}$, etc.

SIFT (scale-invariant feature transform): Keypoint detection

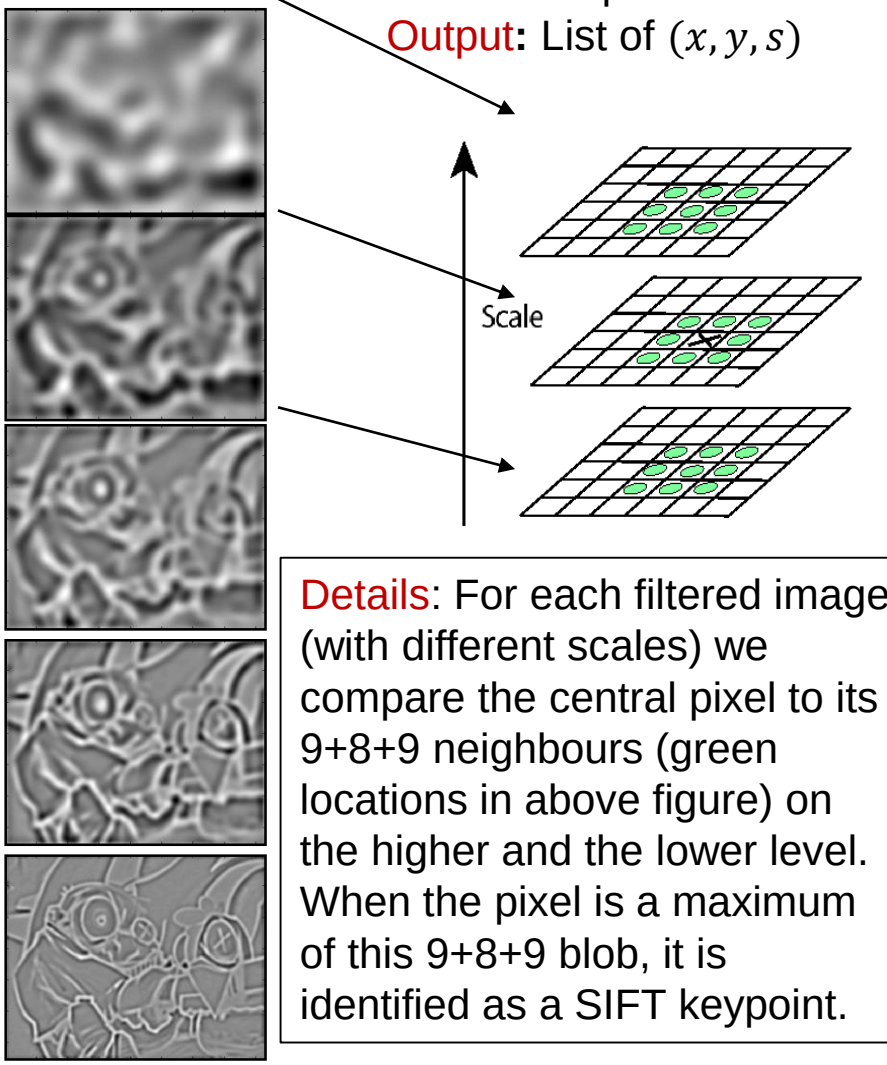
The step-by-step tutorial is available at http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

Image pyramid with Gaussian filter ($k^s\sigma$) for s-th scale, σ is used in Gaussian filter, k is a user-defined hyper-parameter.



Find maxima across scales and positions

Output: List of (x, y, s)



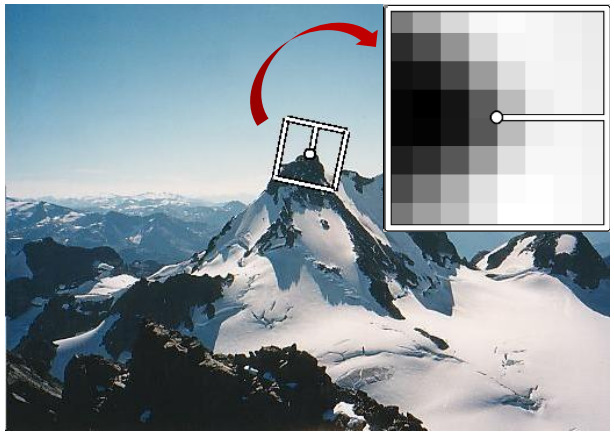
Details: For each filtered image (with different scales) we compare the central pixel to its 9+8+9 neighbours (green locations in above figure) on the higher and the lower level. When the pixel is a maximum of this 9+8+9 blob, it is identified as a SIFT keypoint.

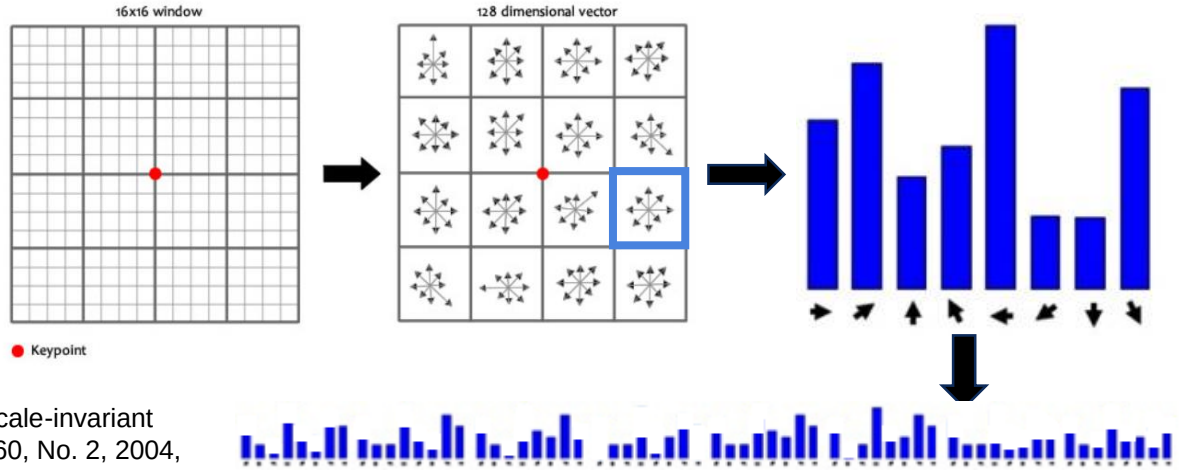
SIFT (scale-invariant feature transform): Feature extraction

version:4.4.0 *July, 2020*

SIFT (Scale-Invariant Feature Transform) algorithm has been moved to the main repository (patent on SIFT is expired)

- Detect keypoints (see previous slide). For each keypoint (at specific location x, y and scale s), warp the region (in the feature map at that specific scale, not the original image) around it to canonical orientation and resize the region to 16×16 pixels. Create histogram of local gradient directions. Assign canonical orientation at peak of the histogram. Rotate the patch so that the dominant orientation points rightward. This makes the patches rotation invariant.
- [Suggested by the original paper] Divide the region into 4×4 squares (totally 16). Each square has 4×4 pixels. For each square, compute gradients for each pixels, then compute gradient direction histogram over 8 directions (bins). Concatenate the histograms computed from 16 squares to obtain a 128 ($16 \times 8 = 128$) dimensional feature.





Reference: D. G. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision, Vol. 60, No. 2, 2004, pp. 91-110

Feature matching using similarity

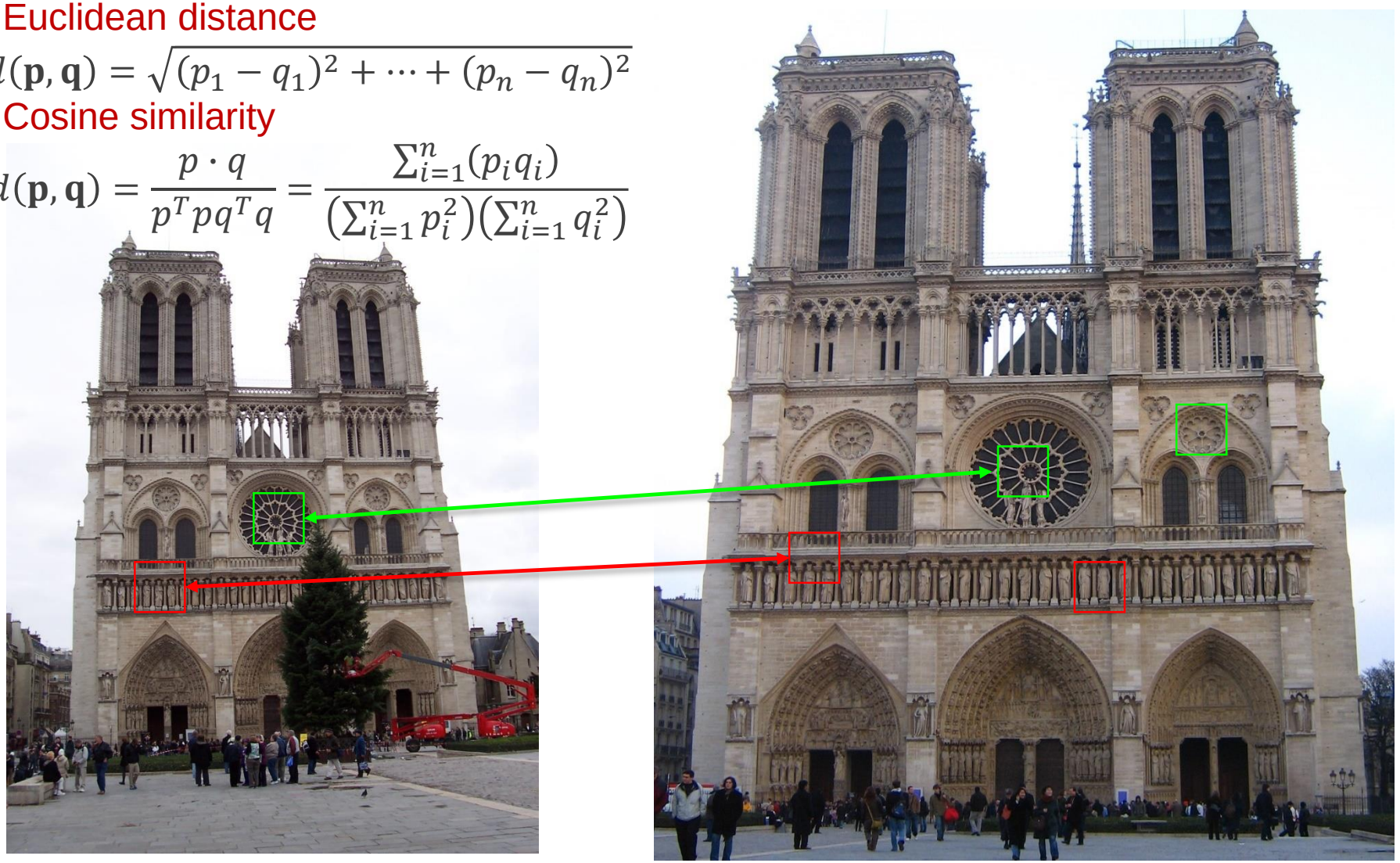
Given features $\mathbf{p}$ and $\mathbf{q}$ that are illustrated as squares in left/right images, respectively.

- Euclidean distance

$$d(\mathbf{p},\mathbf{q}) = \sqrt{(p_1 - q_1)^2 + \cdots + (p_n - q_n)^2}$$

- Cosine similarity

$$d(\mathbf{p},\mathbf{q}) = \frac{p \cdot q}{p^T p q^T q} = \frac{\sum_{i=1}^{n}(p_i q_i)}{\left(\sum_{i=1}^{n} p_i^2\right)\left(\sum_{i=1}^{n} q_i^2\right)}$$



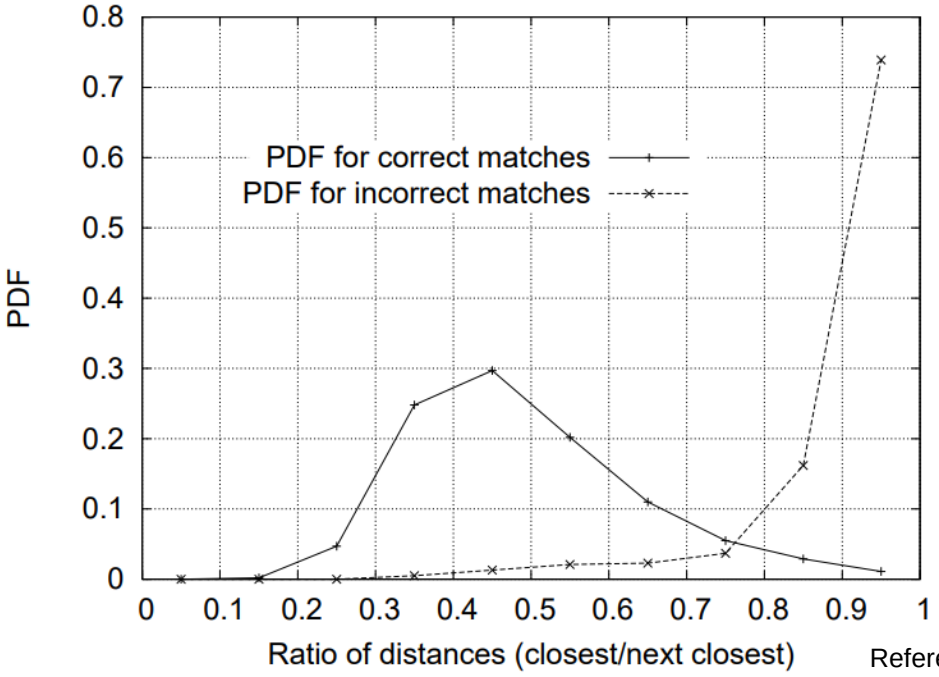
Feature matching using similarity

- Compare distance of *the closest (NN1)* and the *second-closest (NN2)* feature vector neighbor.

$$\text{If } NN1 \approx NN2 \quad Ratio\ \frac{NN1}{NN2} \approx 1 \quad \text{Ambiguity matches}$$

$$\text{If } NN1 \ll NN2 \quad Ratio\ \frac{NN1}{NN2} \to 0 \quad \text{Good match}$$

- Sort matches in the order of this ratio, then choose a threshold to select the reliable matches.



The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the *probability density function* of this ratio for correct matches, while the dotted line is for matches that were incorrect.

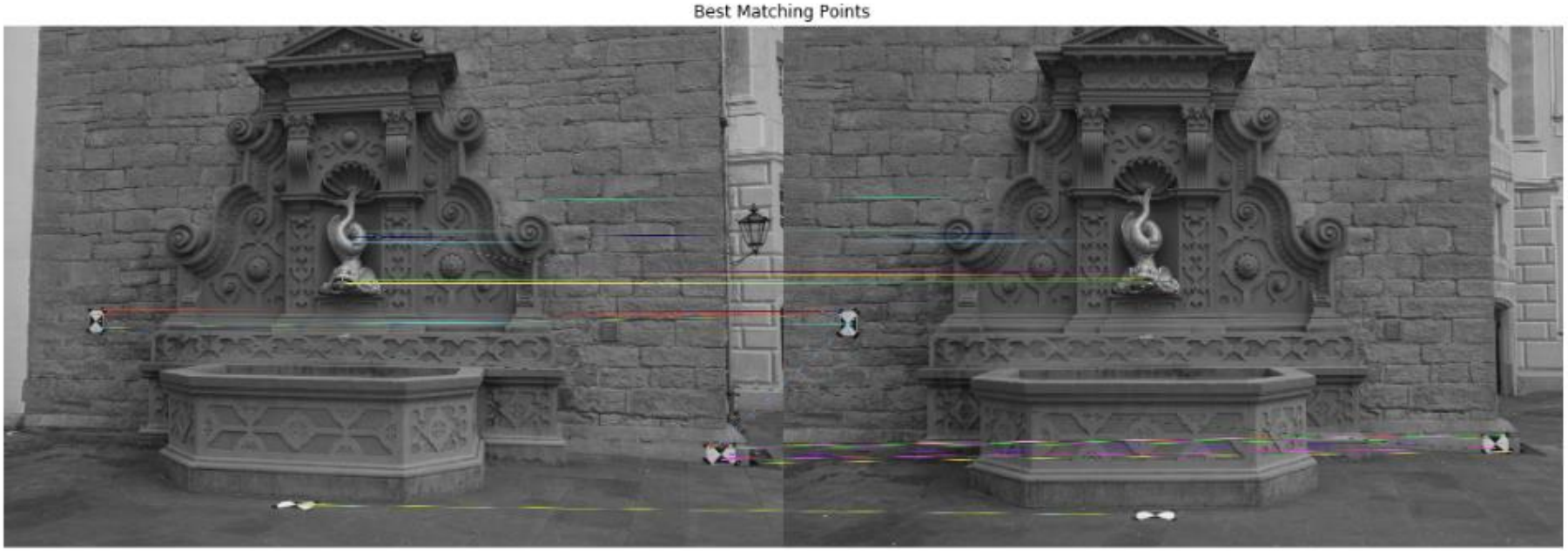
Reference: D. G. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision, 60, 2, 2004, pp. 91-110.

Discussion

Question: SIFT can find the point-to-point matching from two images and their point-to-point similarities. How can we determine the similarity between two IMAGES?

Workshop 3D sensor data representation and modelling

- Task: Feature extraction and matching from multiple view images
- Dataset: SfM Camera trajectory quality evaluation, https://github.com/openMVG/SfM_quality_evaluation



Thank you!

Dr TIAN Jing
Email: tianjing@nus.edu.sg