

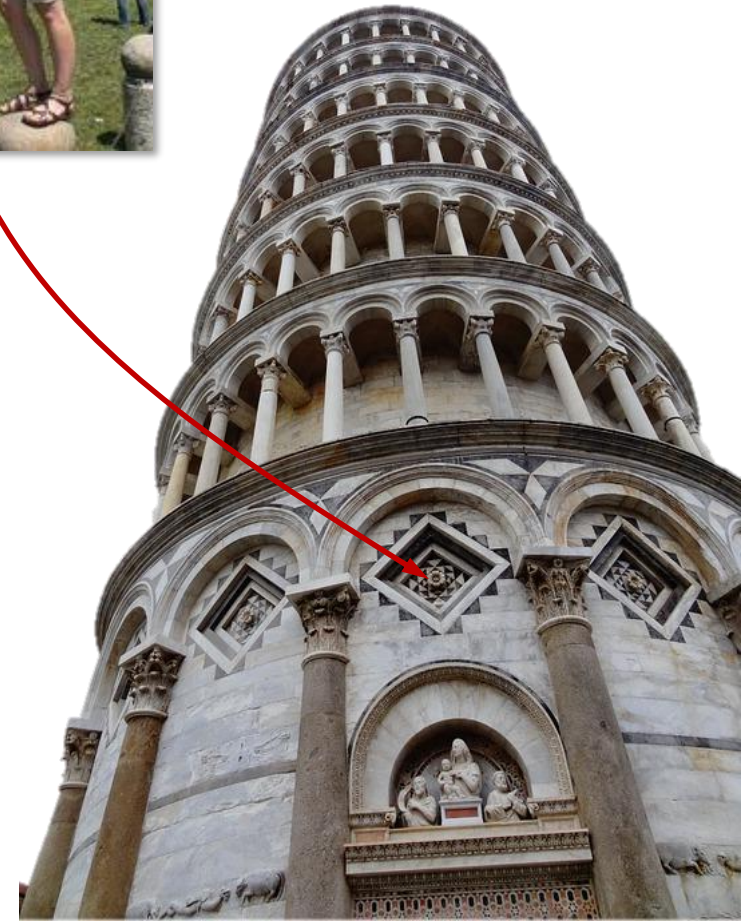
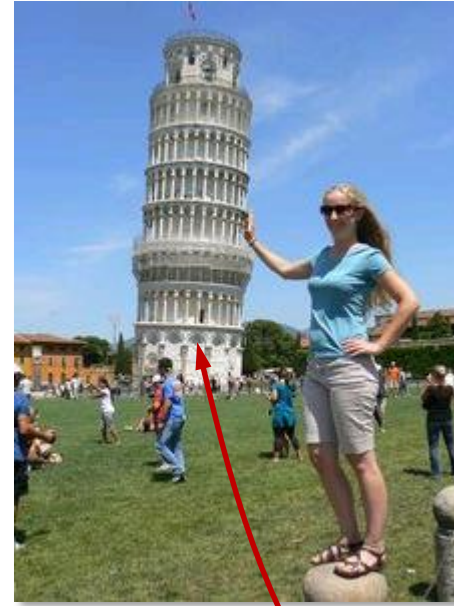


An Invitation to 3D Vision: Finding Correspondence

Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | <https://mint-lab.github.io/>

Table of Contents: Finding Correspondence

- **Feature Points**
 - Q) How can we detect salient points (or parts)?
- **Feature Descriptors**
 - Q) How can we distinguish the points each other?
- **Feature Matching**
 - Q) How can we associate the points across different images?
- **Feature Tracking**
 - Q) How can we associate the points across their next image?
- **Outlier Rejection**
 - Q) How can we select correctly matched points?



Getting Started with a Quiz

- Q) What is it?



600 pixels

Getting Started with a Quiz

- Q) What is it?



Getting Started with a Quiz

- Q) What is it? **"Duck"**



600 pixels



1095 pixels



600 pixels

- Q) Why corners (junction) instead of edges or blobs?
 - Human visual systems understand objects better from corners than edges.
 - a.k.a. feature points, keypoints, salient points, and interest points

Getting Started with a Quiz

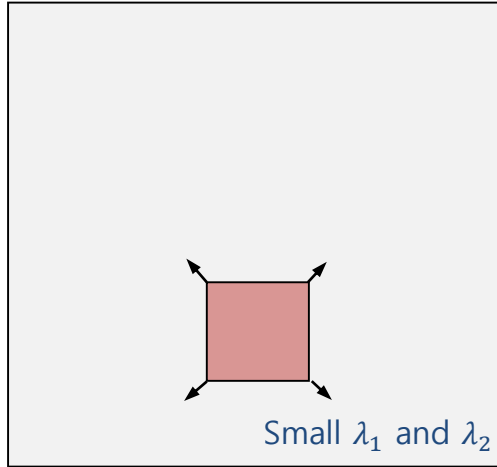
- Q) Where is it?



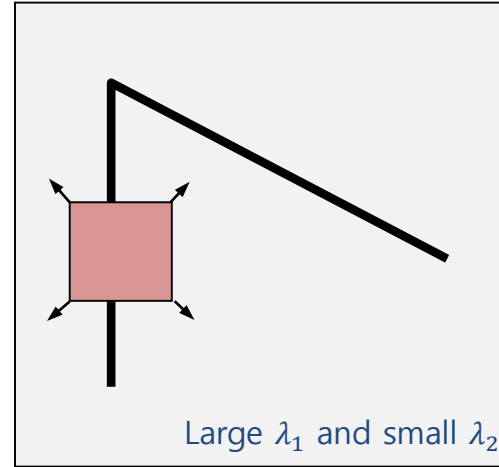
- Q) What is a good feature? (requirements of visual features)
 - **Repeatability** (invariance/robustness to transformation and noise)
 - **Distinctiveness** (easy to distinguish or match)
 - **Locality** (due to occlusion)
 - Quantity, accuracy (localization), efficiency (computing time), ...

Feature Point) Harris Corner (1988)

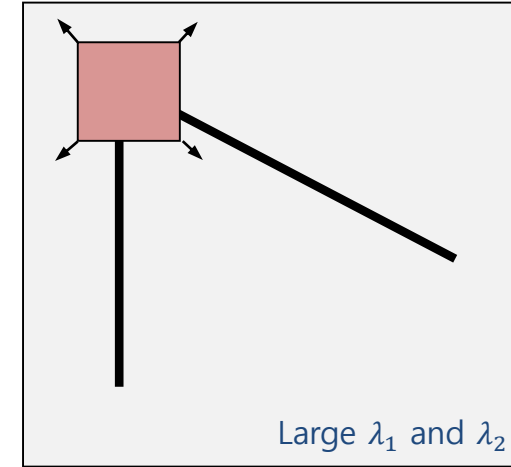
- Key idea: **Sliding window**



"flat" region:
no change
in all directions



"edge":
no change
along the edge direction



"corner":
significant change
in all directions

– Formulation

- $I(x + \Delta_x, y + \Delta_y) \approx I(x, y) + [I_x(x, y) \quad I_y(x, y)] \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$ where $I_x = \frac{\partial I}{\partial x}$ and $I_y = \frac{\partial I}{\partial y}$
- $D(\Delta_x, \Delta_y) = \sum_{(x,y) \in W} (I(x + \Delta_x, y + \Delta_y) - I(x, y))^2 \approx [\Delta_x \quad \Delta_y] \underbrace{\begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix}}_M [\Delta_x \\ \Delta_y]$

M

Feature Point) Harris Corner (1988)

- Key idea: **Sliding window**

- Formulation

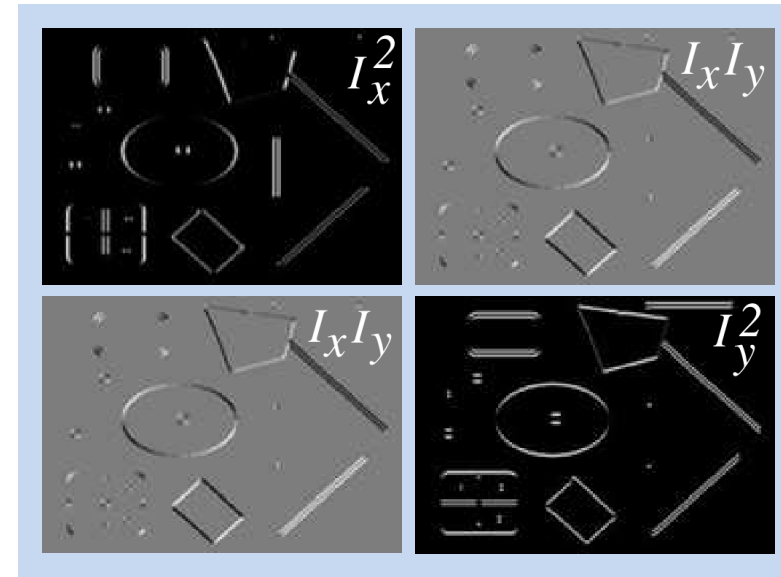
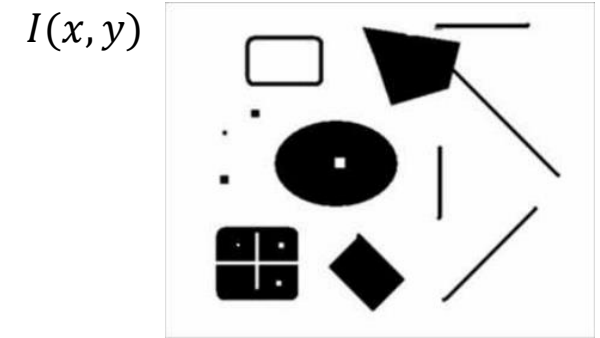
- $I(x + \Delta_x, y + \Delta_y) \approx I(x, y) + [I_x(x, y) \ I_y(x, y)] \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$ where $I_x = \frac{\partial I}{\partial x}$ and $I_y = \frac{\partial I}{\partial y}$
 - $D(\Delta_x, \Delta_y) = \sum_{(x,y) \in W} (I(x + \Delta_x, y + \Delta_y) - I(x, y))^2 \approx [\Delta_x \ \Delta_y] \underbrace{\begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix}}_M [\Delta_x \ \Delta_y]$

- **Harris corner response**

- $\text{cornerness} = \det(M) - k \text{trace}(M)^2$
 - Note) $\det(M) = \lambda_1 \lambda_2$, $\text{trace}(M) = \lambda_1 + \lambda_2$, and $k \in [0.04, 0.06]$

- Note) **Good-Feature-to-Track** (a.k.a. GFTT or Shi-Tomasi corner; 1994)

- $\text{cornerness} = \min(\lambda_1, \lambda_2)$



M

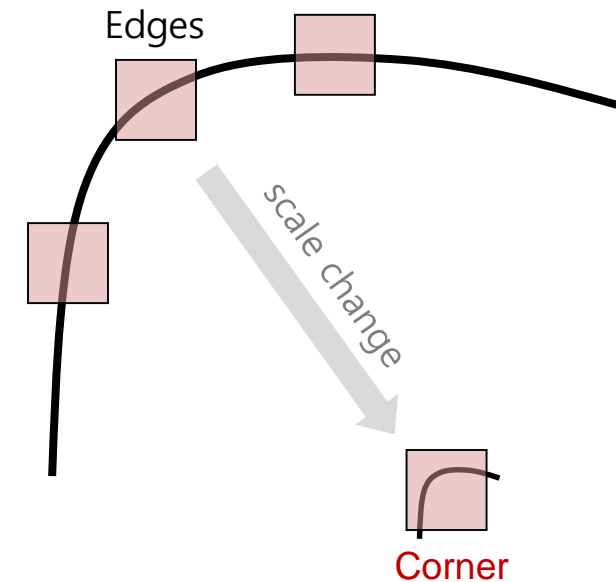


$\det(M) - k \text{trace}(M)^2$

Feature Point) Harris Corner (1988)

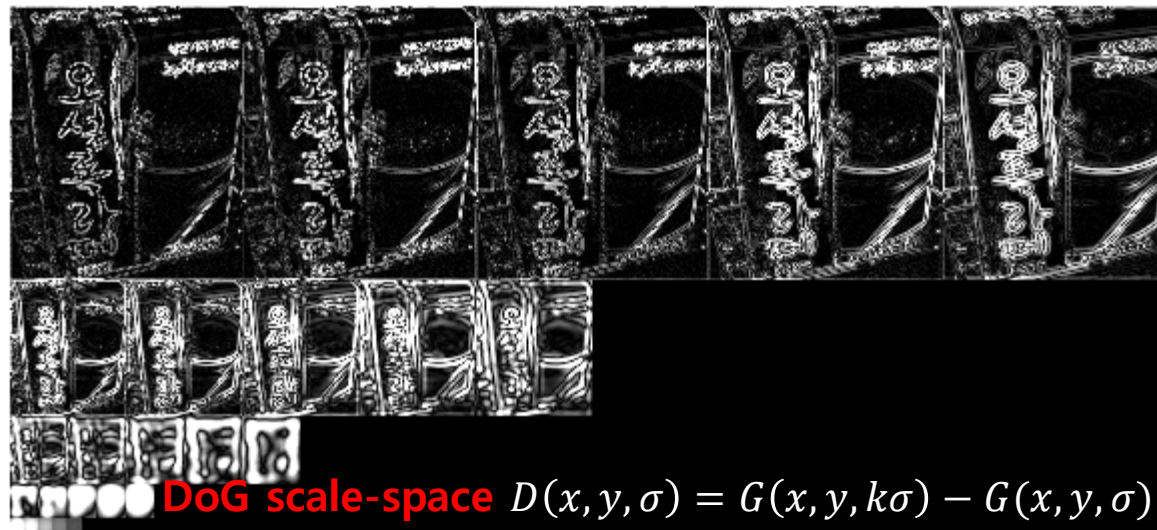
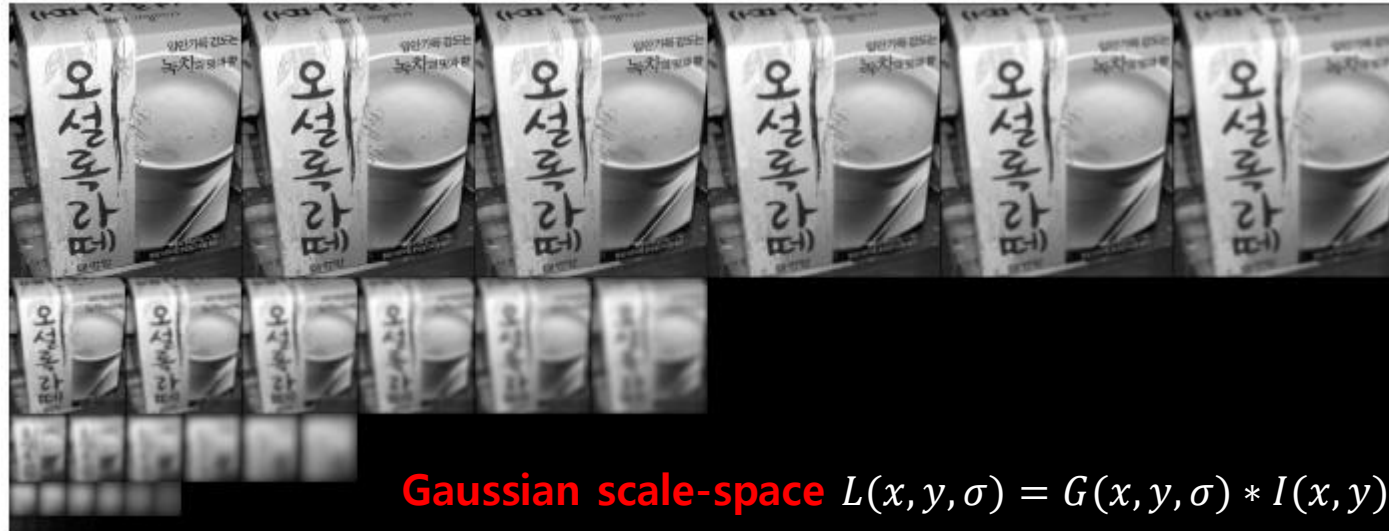
- Properties

- Invariant to translation, rotation, and intensity shift ($I \rightarrow I + b$) ~~intensity scaling ($I \rightarrow aI$)~~
- Variant to **image scaling**



Feature Point) SIFT (Scale-Invariant Feature Transform; 1999)

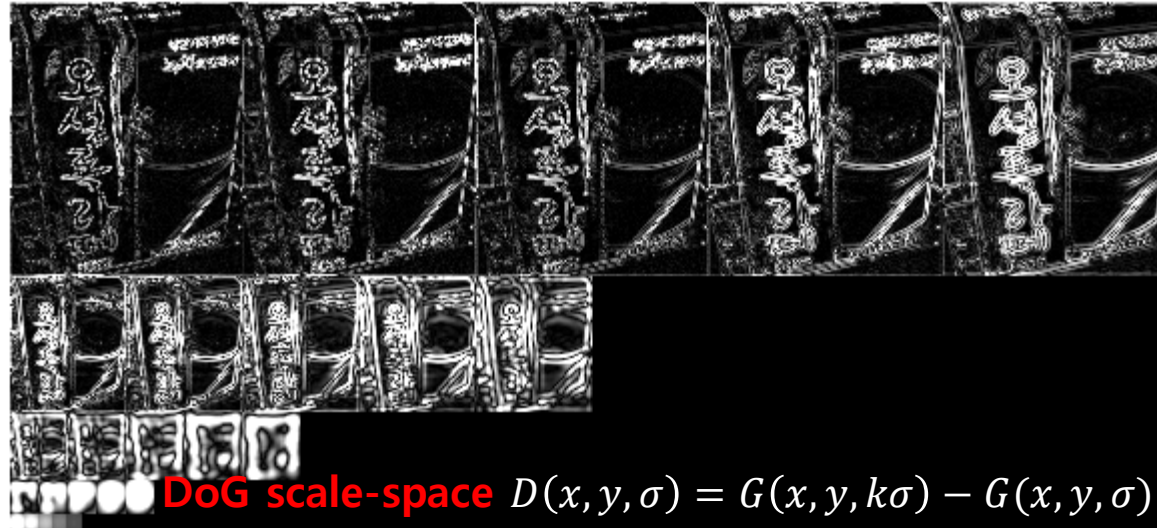
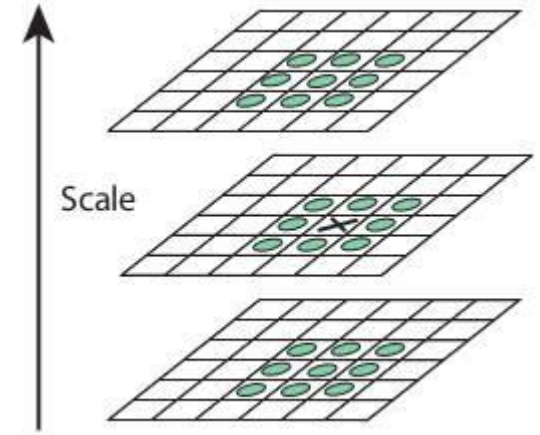
- Key idea: **Scale-space** (~ [image pyramid](#)) and **DoG** ([difference of Gaussian](#))



Feature Point) SIFT (Scale-Invariant Feature Transform; 1999)

- Key idea: **Scale-space** (~ [image pyramid](#)) and **DoG** ([difference of Gaussian](#))
- Part #1) **Feature point detection**
 1. Find **local extrema** (minima and maxima) in DoG scale-space
 2. Localize their position accurately (sub-pixel level) using 3D quadratic function
 3. Eliminate **low contrast candidates**, $|D(\mathbf{x})| < \tau$
 4. Eliminate **candidates on edges**,

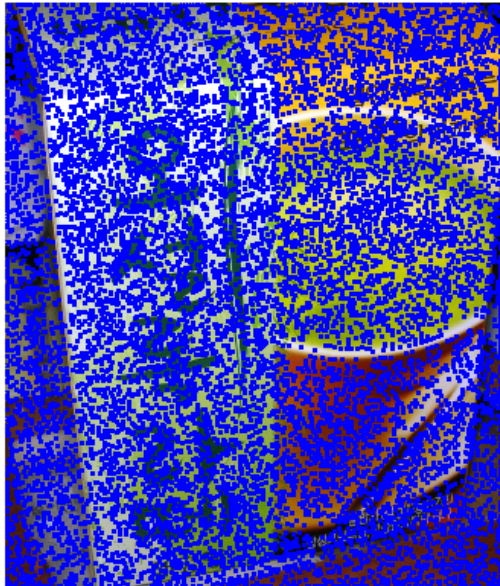
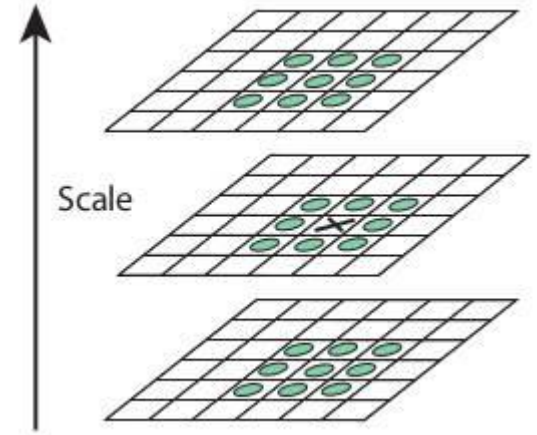
$$\frac{\text{trace}(H)^2}{\det(H)} < \frac{(r+1)^2}{r} \quad \text{where} \quad H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$



Feature Point) SIFT (Scale-Invariant Feature Transform; 1999)

- Key idea: **Scale-space** (~ [image pyramid](#)) and **DoG** ([difference of Gaussian](#))
- Part #1) **Feature point detection**
 1. Find **local extrema** (minima and maxima) in DoG scale-space
 2. Localize their position accurately (sub-pixel level) using 3D quadratic function
 3. Eliminate **low contrast candidates**, $|D(\mathbf{x})| < \tau$
 4. Eliminate **candidates on edges**,

$$\frac{\text{trace}(H)^2}{\det(H)} < \frac{(r+1)^2}{r} \quad \text{where} \quad H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$



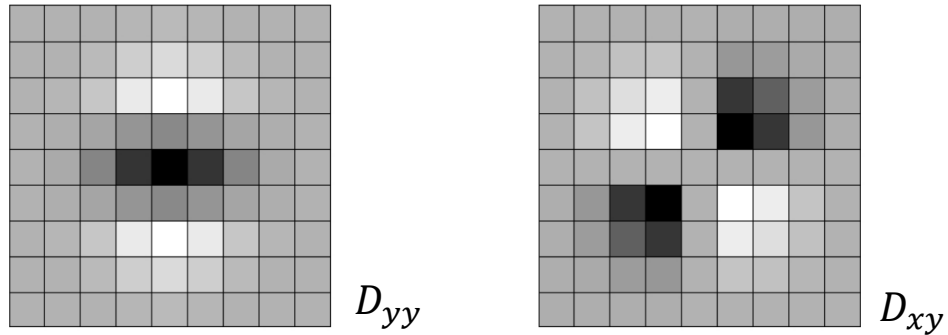
Local extrema (N: 11479)



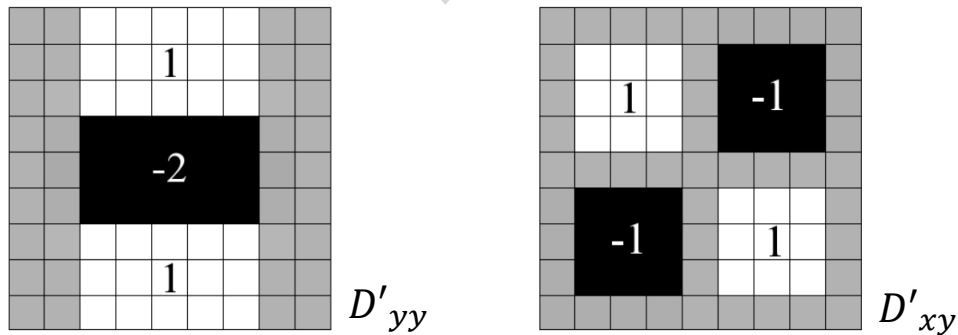
Feature points (N: 971)

Feature Point) SURF (Speeded Up Robust Features; 2006)

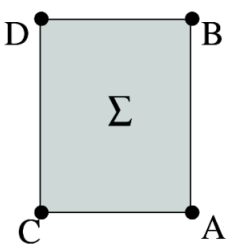
- Key idea: **Approximation of SIFT**
 - e.g. DoG approximation Haar-like features and **integral image**



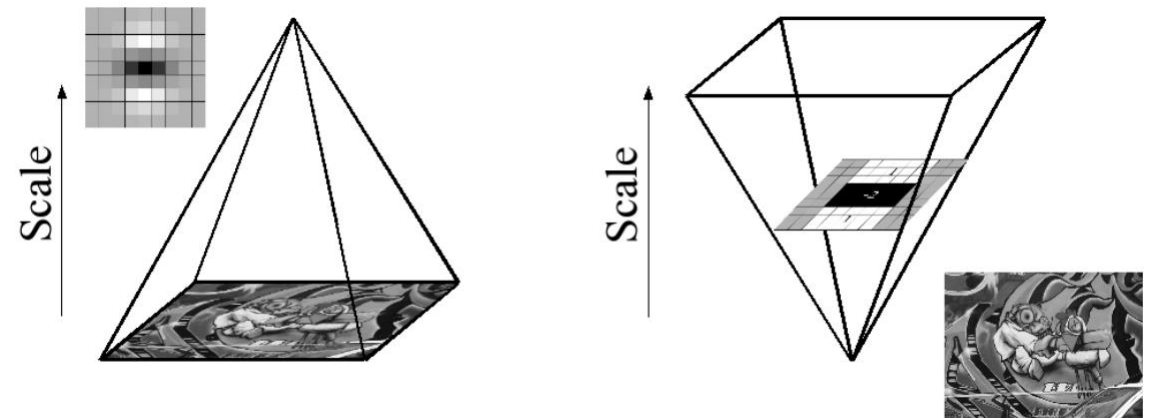
DoG approximation



O

$$S(x, y) = \sum_{i=0}^x \sum_{j=0}^y I(i, j)$$


$\Sigma = A - B - C + D$



Feature Descriptor) SIFT (Scale-Invariant Feature Transform; 1999)

■ Part #2) Orientation assignment

1. Derive magnitude and orientation of gradient of each patch

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

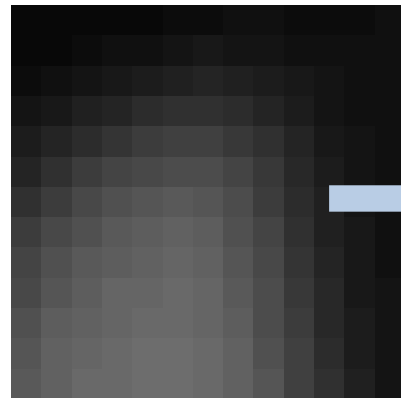
$$\theta(x, y) = \tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$$

2. Find **the strongest orientation**

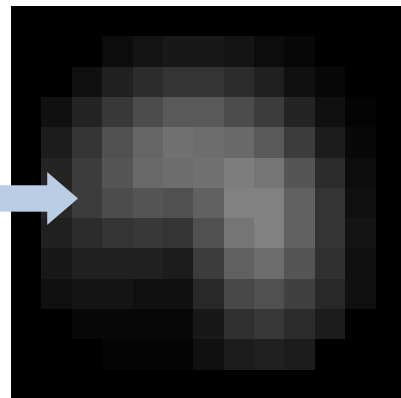
- Histogram voting (36 bins) with Gaussian-weighted magnitude



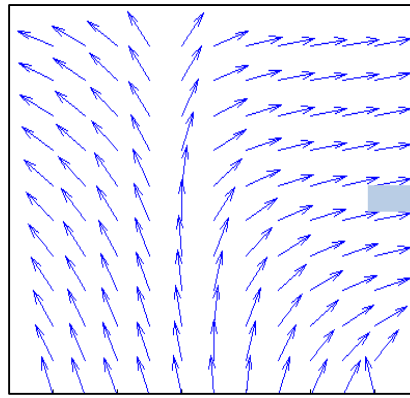
Feature scales and orientations



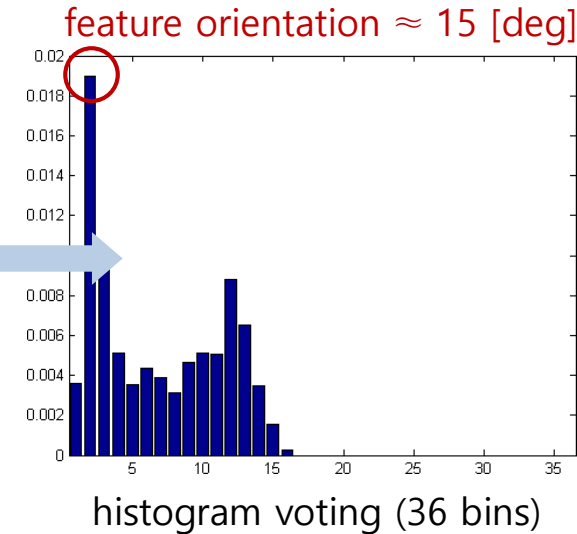
feature patch



gradient magnitude
 $m(x, y)$



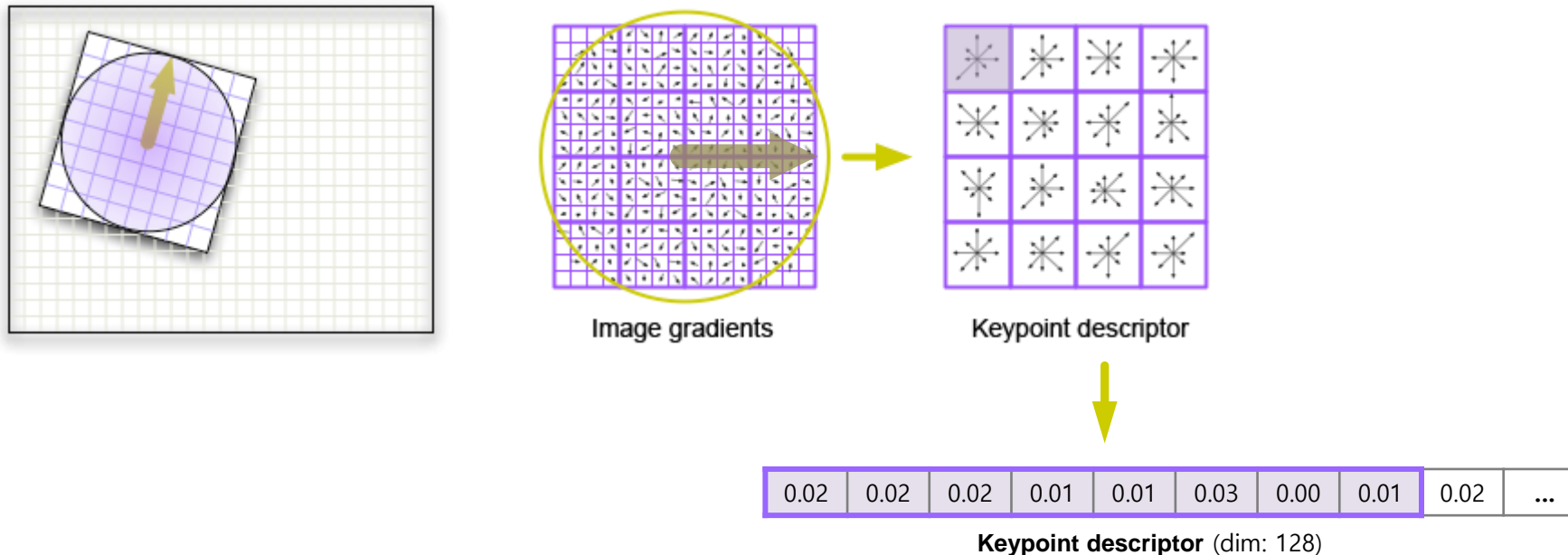
gradient orientation
 $\theta(x, y)$



Feature Descriptor) SIFT (Scale-Invariant Feature Transform; 1999)

■ Part #3) Feature descriptor extraction

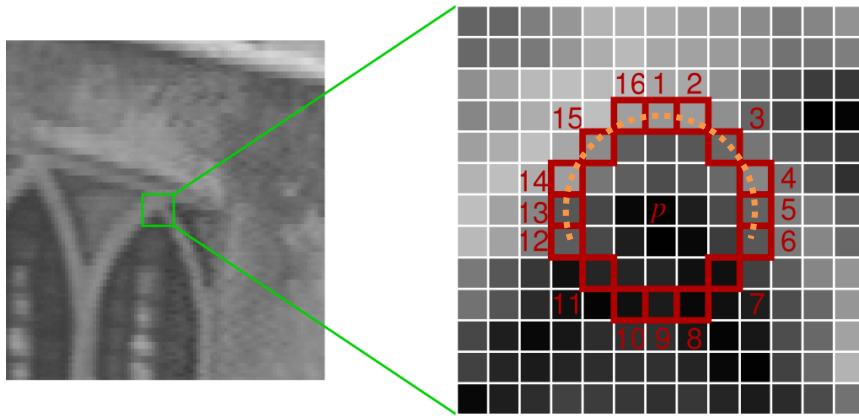
1. Build a 4x4 gradient histogram (8 bins) from each patch (16x16 pixels)
 - Use Gaussian-weighted magnitude again
 - Use relative angles w.r.t. the assigned feature orientation
2. Encode the histogram into a 128-dimensional vector
 - Apply normalization to be an unit vector



Feature scales and orientations

Feature Point) FAST (Features from Accelerated Segment Test; 2006)

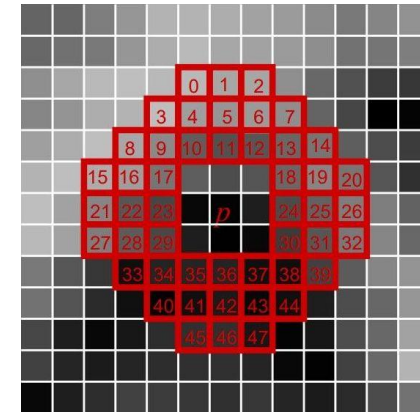
- Key idea: **Intensity check of continuous arc of n pixels**
 - Is this point p a corner? (I_p : intensity at p , t : the intensity threshold)
 - Is a segment of n continuous pixels brighter than $I_p + t$? (OR) Is the segment darker than $I_p - t$?
 - Note) High-speed non-corner rejection: Checking intensity values at 1, 9, 5, and 13 (n : 12)



- Too many corners! → Non-maximum suppression

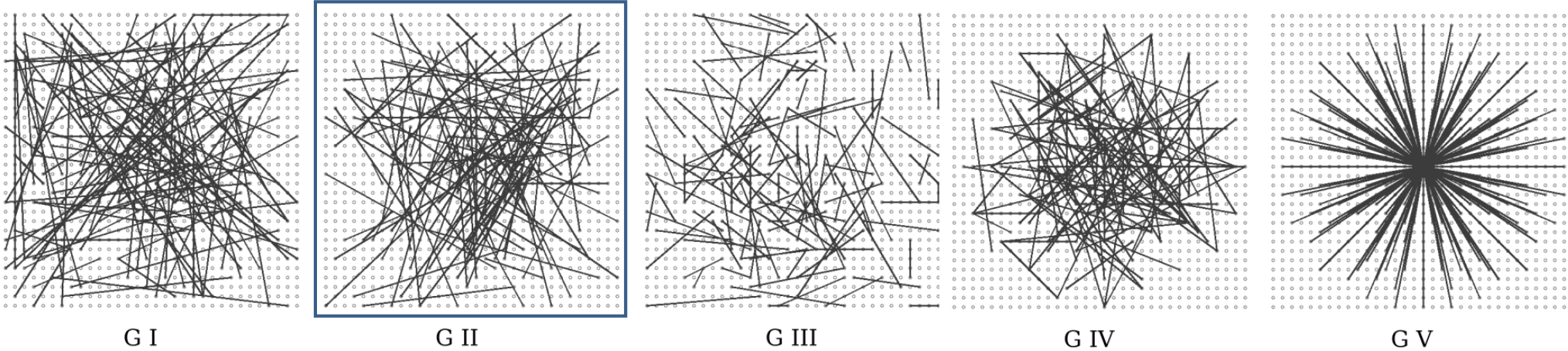
▪ Versions

- FAST-9 (n : 9; `cv.FastFeatureDetector_TYPE_9_16`), FAST-12 (n : 12), ...
- FAST-ER: Training a decision tree to enhance repeatability with more pixels



Feature Descriptor) BRIEF (Binary Robust Independent Elementary Features; 2010)

- Key idea: **Intensity comparison of a sequence of random pairs (binary test)**
 - Path size: 31 x 31 pixels (Note: Applying smoothing for stability and repeatability)



- Descriptor size: 128 tests (128 bits) → 16 bytes
 - Note) SIFT: 128-dimensional vector → 512 bytes
- Versions: The number of tests
 - BRIEF-32, BRIEF-64, BRIEF-128 (16 bytes), BRIEF-256 (32 bytes), BRIEF-512 (64 bytes), ...
- Combination examples
 - SIFT feature points + BRIEF descriptors
 - FAST feature points + BRIEF descriptors

Feature Point and Descriptor) **ORB (Oriented FAST and rotated BRIEF, 2011)**

- Key idea: **Adding rotation invariance to BRIEF**

- **Oriented FAST**

- Generate scale pyramid for scale invariance
 - Detect *FAST-9* points (filtering with Harris corner response)
 - Calculate feature orientation by *intensity centroid* $C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}})$

$$\theta = \tan^{-1} \frac{m_{01}}{m_{10}} \quad \text{where} \quad m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

- **Rotation-aware BRIEF**

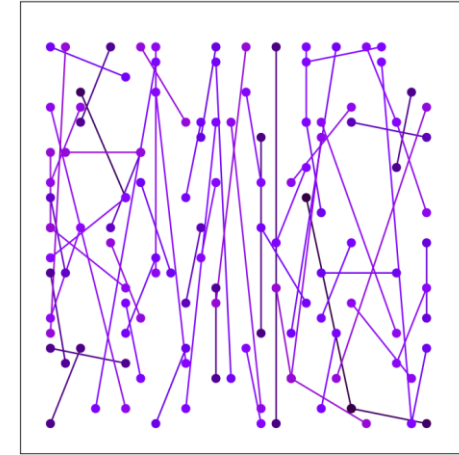
- Extract BRIEF descriptors w.r.t. the known orientation
 - Use better comparison pairs trained by greedy search

- Combination (default): **ORB**

- FAST-9 detector (with orientation) + BRIEF-256 descriptor (with trained pairs)

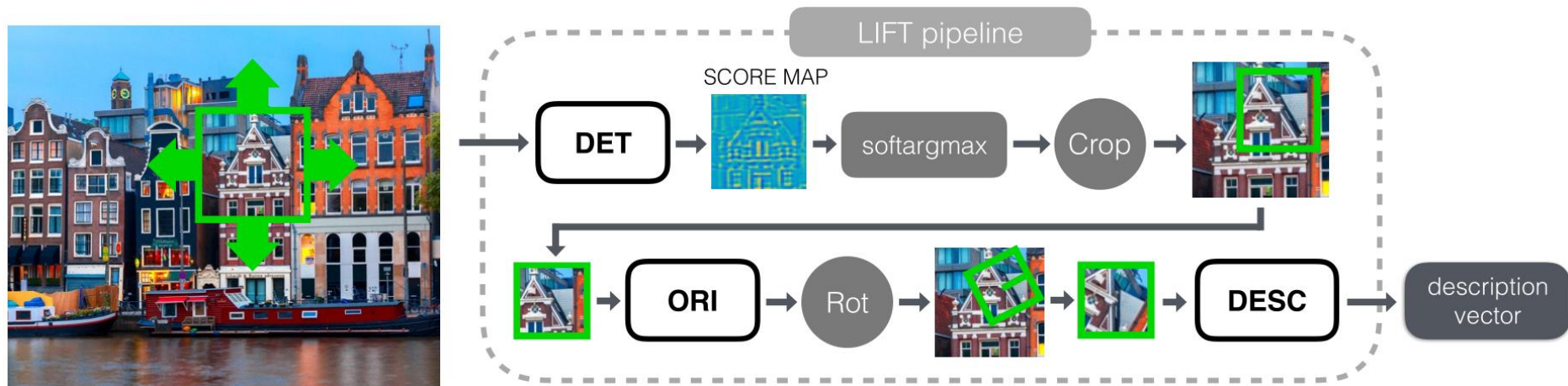
- Computing time

- ORB: **15.3 [msec]** / SURF: 217.3 [msec] / SIFT: 5228.7 [msec] @ 24 images (640x480) in Pascal dataset

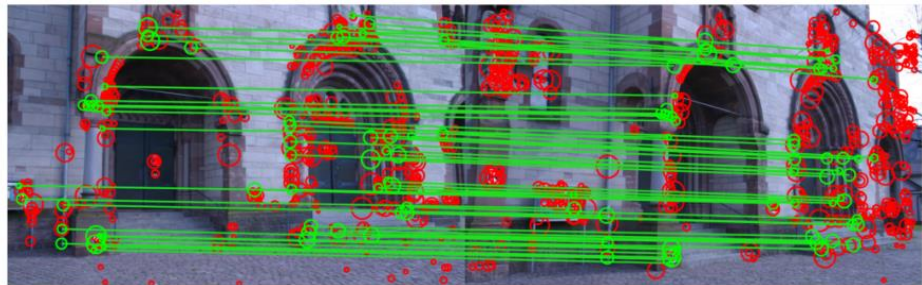


Feature Point and Descriptor) **LIFT (Learned Invariant Feature Transform; 2016)**

- Key idea: **Deep neural network**
 - CNN network: **DET** (feature detector) + **ORI** (orientation estimator) + **DESC** (feature descriptor)
 - Training data: Photo Tourism dataset with [VisualSFM](#) (SIFT)



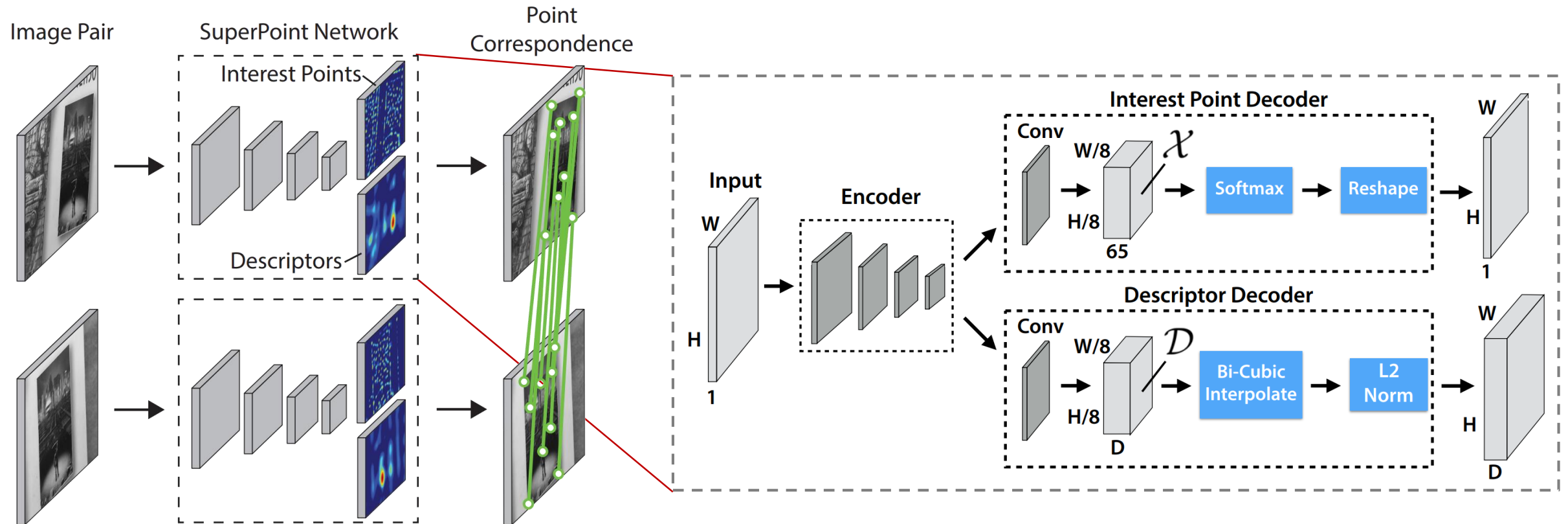
SIFT



LIFT

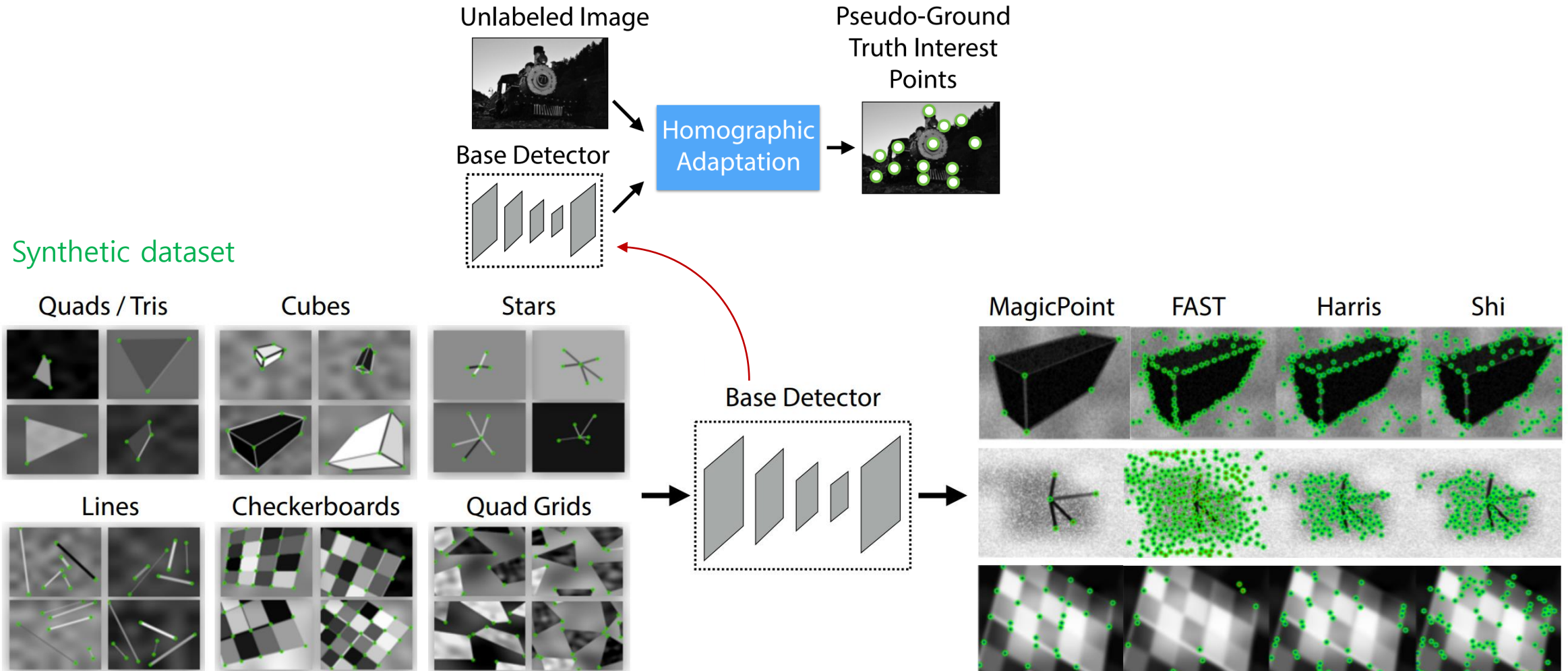
Feature Point and Descriptor) SuperPoint (2020)

- Key idea: **Self-supervised training with homography transformation**
 - CNN network: Encoder (~ VGG) + Decoders (for point and descriptor)



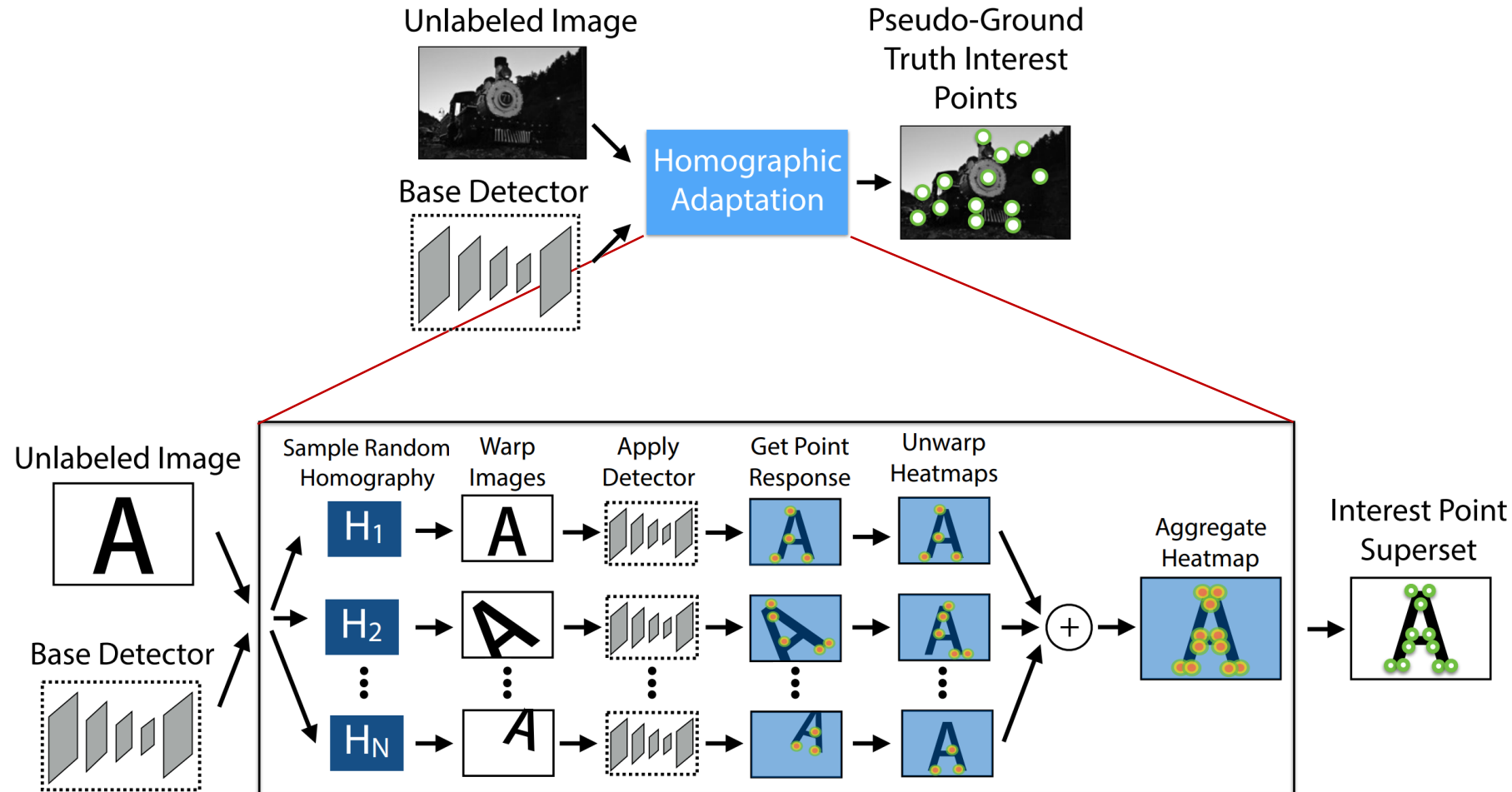
Feature Point and Descriptor) **SuperPoint (2020)**

- Key idea: **Self-supervised training with homography transformation**
 - Training data generation: The base detector, *MagicPoint* → Ground truth interest points



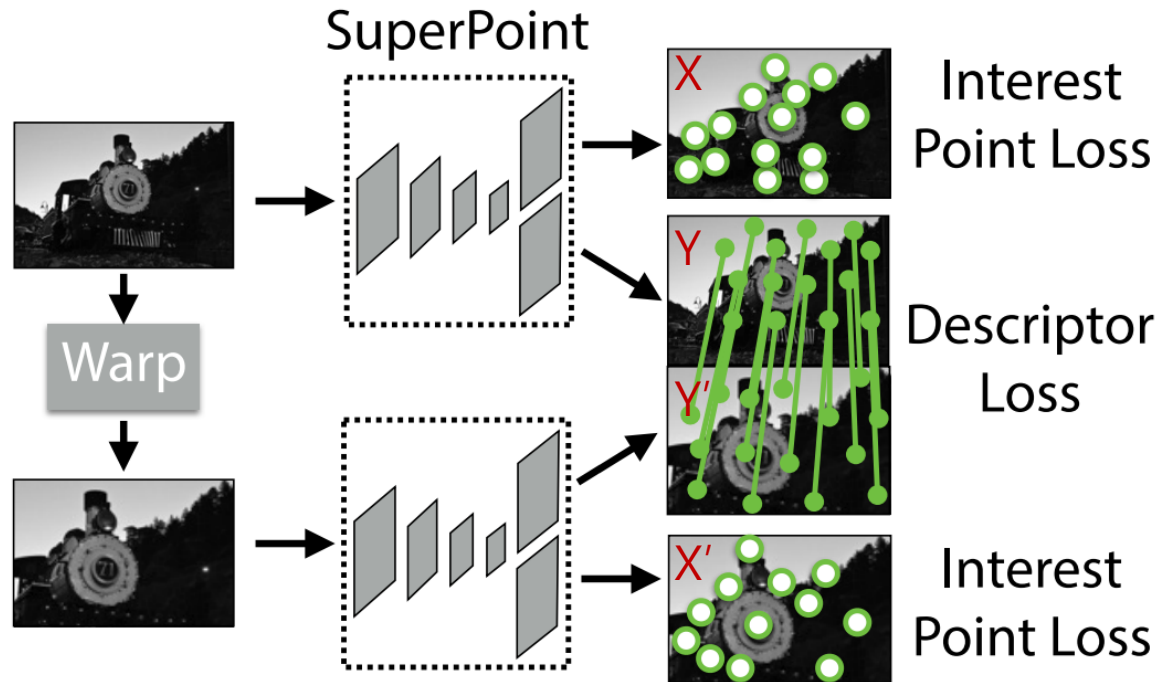
Feature Point and Descriptor) **SuperPoint (2020)**

- Key idea: **Self-supervised training with homography transformation**
 - Training data generation: The base detector, *MagicPoint* → Ground truth interest points



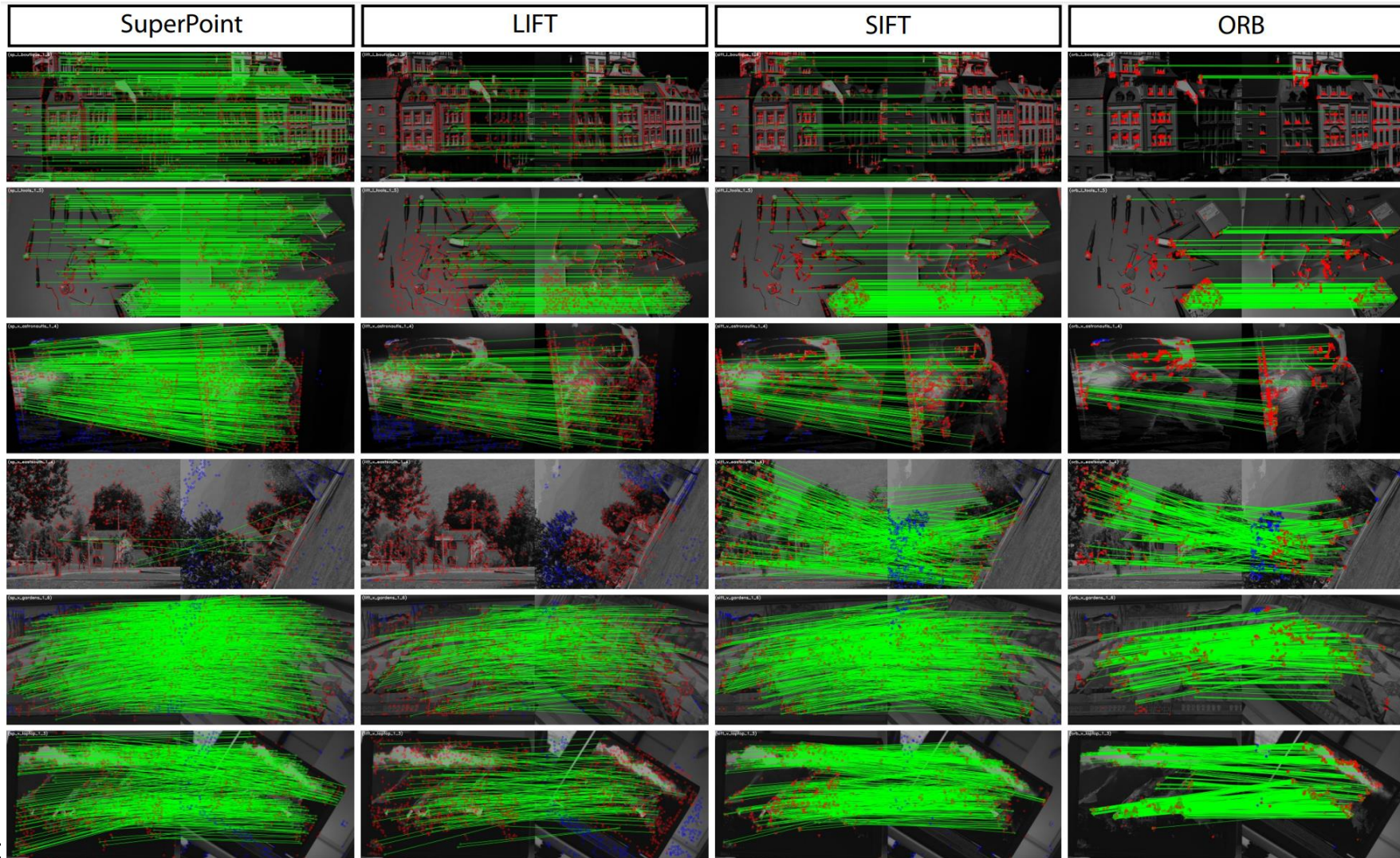
Feature Point and Descriptor) SuperPoint (2020)

- Key idea: **Self-supervised training with homography transformation**
 - Training data augmentation: Random homography transformation
 - Loss functions: Interest Point Loss (X, Y) + Interest Point Loss (X', Y') + Descriptor Loss (Y, Y')



Feature Point and Descriptor) SuperPoint (2020)

- Key idea: **Self-supervised training with homography transformation**
 - Real-time performance: **70 FPS** (13 msec) on 480 x 640 images with NVIDIA Titan X GPU



Summary) Feature Points and Descriptors

Feature Points	Gradient-based <ul style="list-style-type: none">▪ Harris▪ GFTT (a.k.a. Shi-Tomasi)▪ SIFT▪ SURF	Intensity-based <ul style="list-style-type: none">▪ FAST	DL-based <ul style="list-style-type: none">▪ LIFT▪ SuperPoint
Feature Descriptor	Real-valued <ul style="list-style-type: none">▪ SIFT▪ SURF	Binary-valued <ul style="list-style-type: none">▪ BRIEF▪ ORB (FAST+BRIEF)	DL-based (Real-valued) <ul style="list-style-type: none">▪ LIFT▪ SuperPoint
Advantages Disadvantages	(+) Accurate (-) Slow	(+) Fast (-) Inaccurate (+) Less storage	(+) Accurate (+) Fast (-) GPU requirement

Feature Matching) For Real-valued Descriptors

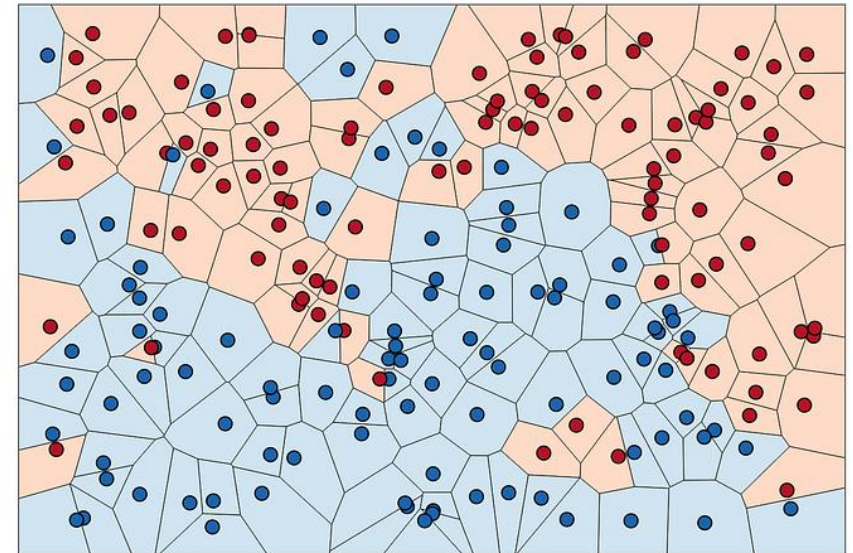
- **Real-valued descriptors**

- Distance measures

- [Euclidean distance](#): $l_2(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_2$
 - [Cosine similarity](#): $s_c(\mathbf{d}, \mathbf{d}') = \frac{\mathbf{d} \cdot \mathbf{d}'}{\|\mathbf{d}\| \|\mathbf{d}'\|}$
 - Note) Matching measures can be combined or advanced.
 - e.g. The ratio of the best and second best similarity > threshold
 - It may select more distinguishable feature matching.

- Matching algorithms

- [Brute-force search](#)
 - Time complexity: $O(N)$ for N descriptors
 - Approximated [nearest neighborhood search](#) (ANN search)
 - Time complexity: $O(\log N)$ or less for N descriptors
 - Note) [Big-ANN Competition](#) (recent: NeurIPS 2023)



Feature Matching) For Binary-valued Descriptors and Image Patches

- **Binary-valued descriptors**

- Distance measures

- [Hamming distance](#): $l_h(\mathbf{d}, \mathbf{d}') = \sum_i (d_i \neq d'_i)$

- e.g. (0110, 1110) = 1 vs. (6, 14)

- e.g. (0110, 0111) = 1 vs. (6, 7)

- e.g. (0110, 0101) = 2 vs. (6, 5)

- Note) Hamming distance is the L_1 -norm with binary-valued descriptors.

- Matching algorithms

- [Brute-force search](#)

- Note) **Raw image patches** can be used as descriptors.

- Distance measure

- SAD ([sum of absolute difference](#)): $l_1(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_1$

- SSD (sum of squared difference): $l_2(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_2$

- ZNCC (zero-mean normalized cross-correlation)

Feature Tracking) Lukas-Kanade Optical Flow (1981)

- Key idea: **Finding movement of a patch whose pixel values are same**

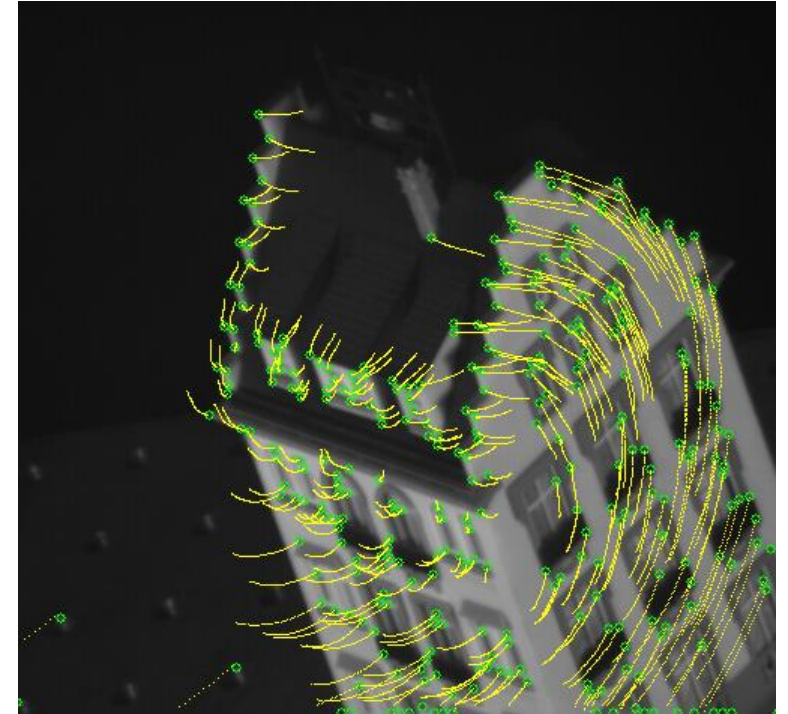
- **Brightness constancy constraint:** $I(x, y, t) = I(x + \Delta_x, y + \Delta_y, t + \Delta_t)$

$$I_x \frac{\Delta_x}{\Delta_t} + I_y \frac{\Delta_y}{\Delta_t} + I_t = 0 \quad \leftarrow \quad I(x + \Delta_x, y + \Delta_y, t + \Delta_t) \approx I(x, y, t) + I_x \Delta_x + I_y \Delta_y + I_t \Delta_t$$

$$A\mathbf{x} = \mathbf{b} \quad \text{where} \quad A = [I_x \quad I_y], \quad \mathbf{x} = [\Delta_x, \Delta_y]^T, \quad \text{and} \quad \mathbf{b} = [-I_t] \quad (\Delta_t = 1)$$

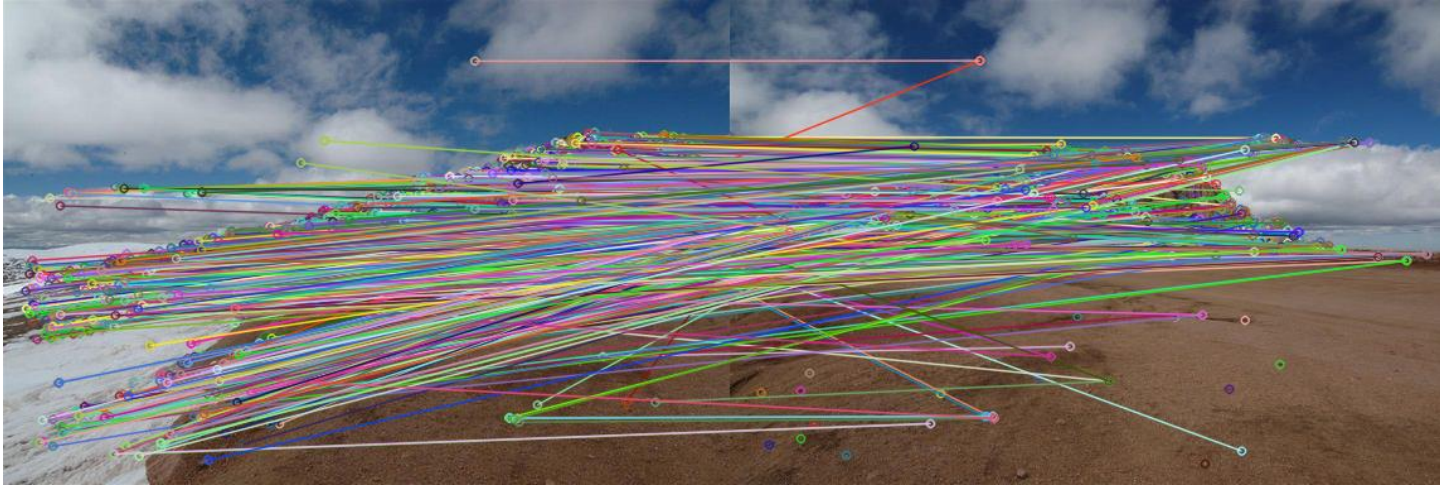
$$\therefore \mathbf{x} = A^+ \mathbf{b}$$

- Combination: **KLT tracker**
 - GFTT detector (a.k.a. Shi-Tomasi) + **Lukas-Kanade** optical flow
- Advantages and disadvantages (feature tracking vs. matching)
 - (+) No descriptor required (\rightarrow fast and compact)
 - (-) Continuous feature tracking causes drift errors.
 - (-) Not working in wide-baseline cases
 - (+) Able to control matching range

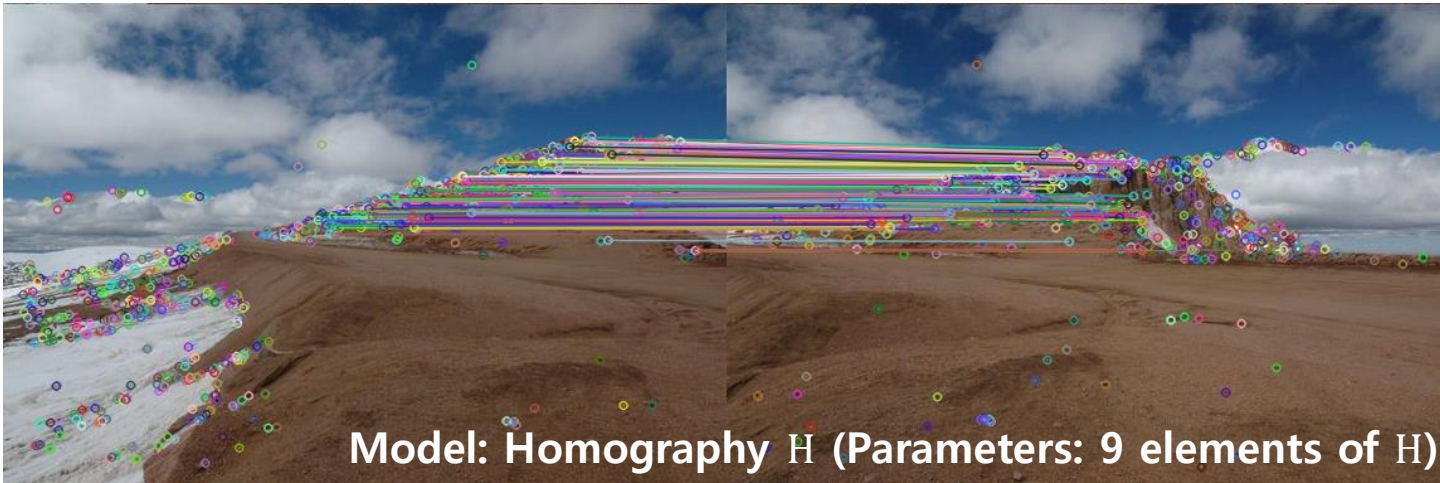


Why Outliers?

Putative feature matches (inliers + outliers)



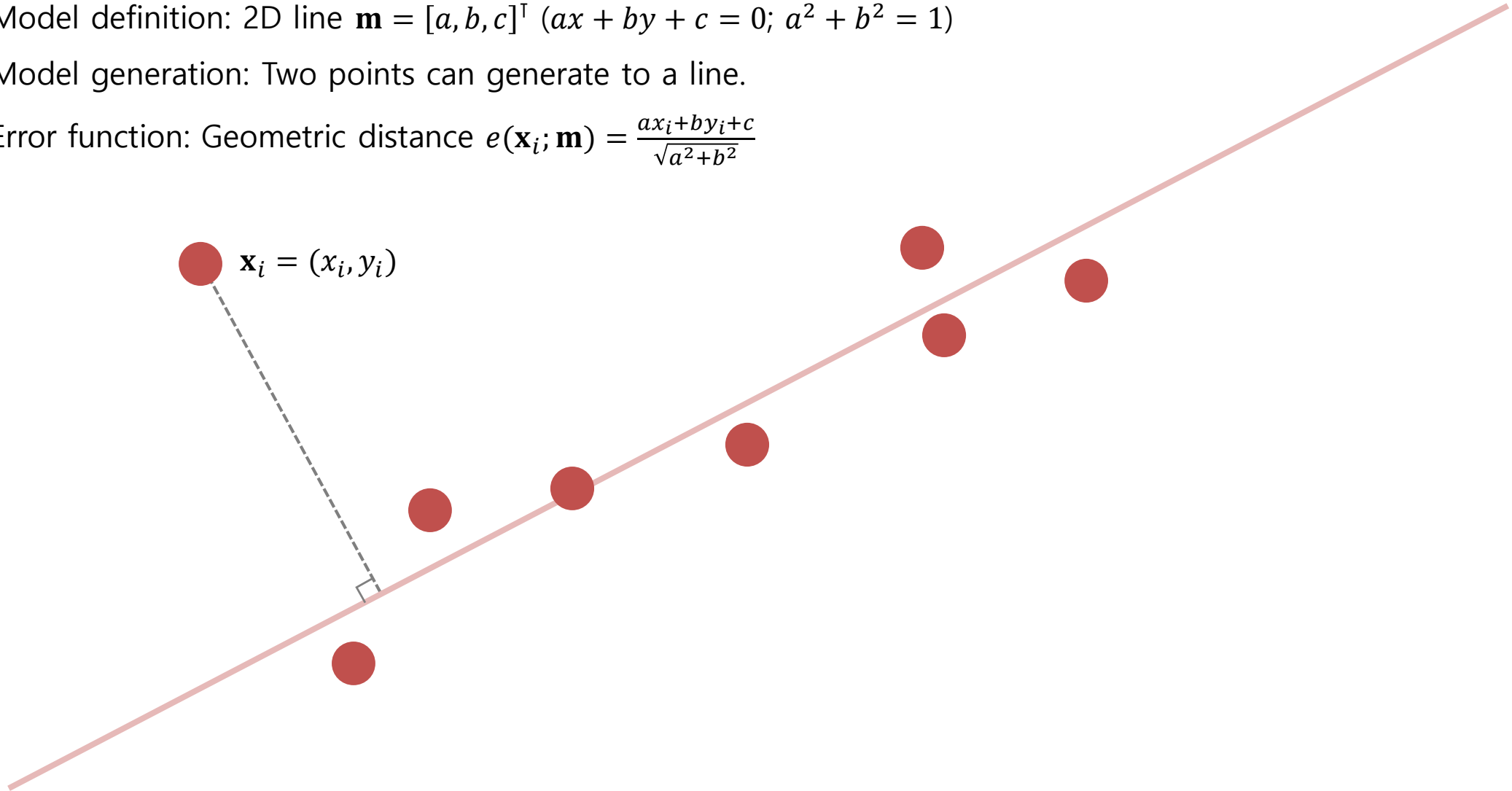
After applying RANSAC (inliers)



Outlier Rejection) RANSAC (Random Sample Consensus; 1981)

- Example) **Line fitting with [RANSAC](#)**

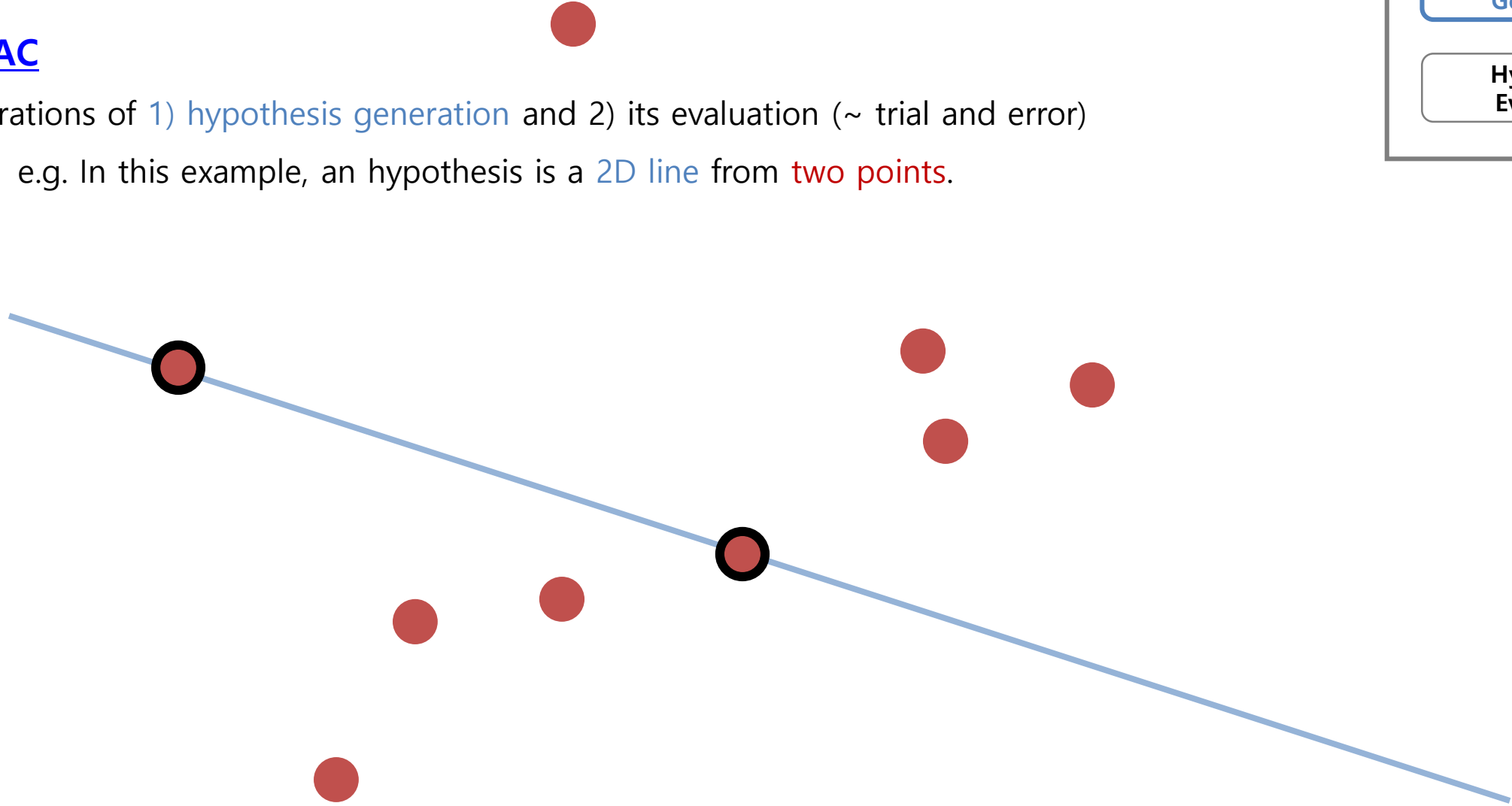
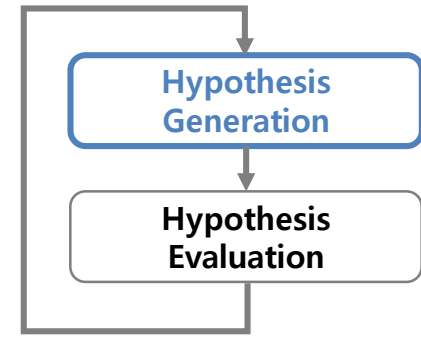
- Model definition: 2D line $\mathbf{m} = [a, b, c]^T$ ($ax + by + c = 0$; $a^2 + b^2 = 1$)
- Model generation: Two points can generate to a line.
- Error function: Geometric distance $e(\mathbf{x}_i; \mathbf{m}) = \frac{ax_i + by_i + c}{\sqrt{a^2 + b^2}}$



Outlier Rejection) RANSAC (Random Sample Consensus; 1981)

- RANSAC

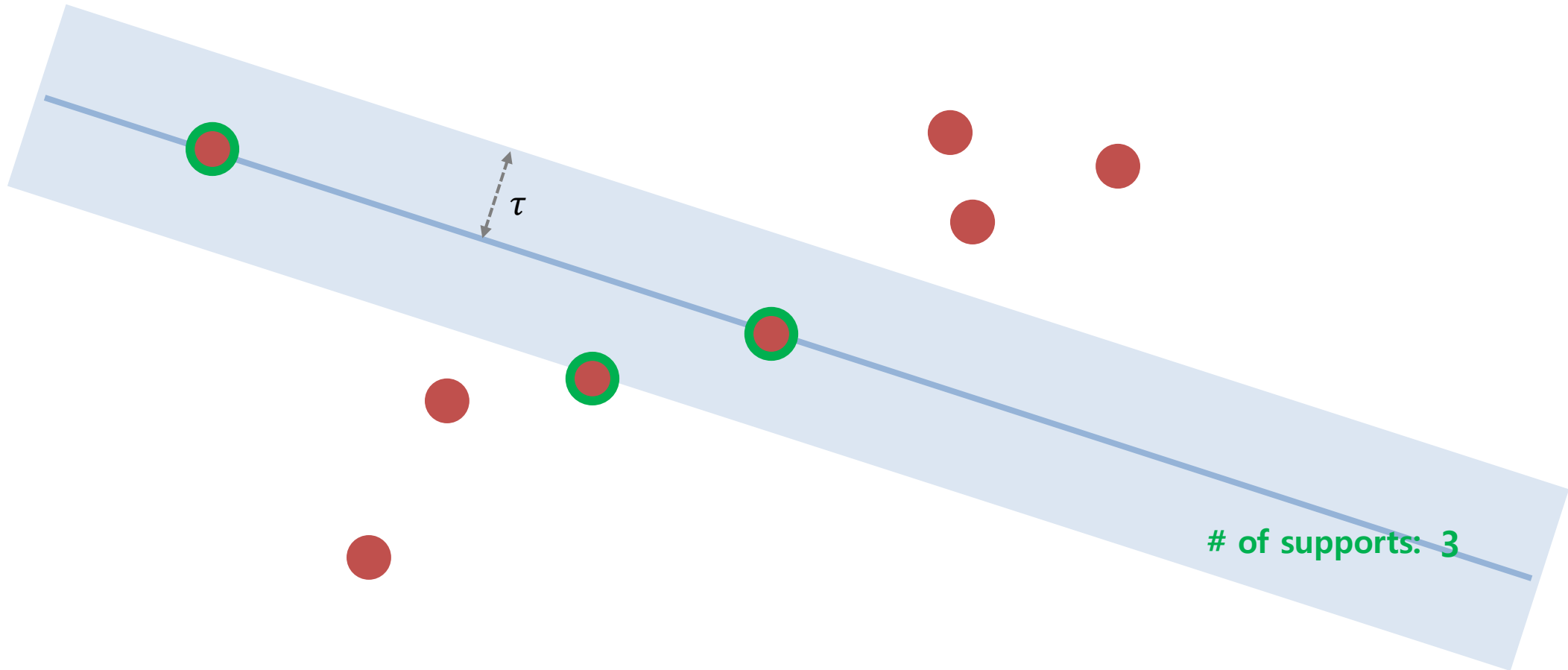
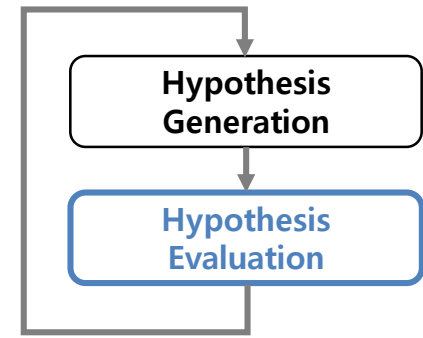
- Iterations of 1) hypothesis generation and 2) its evaluation (~ trial and error)
 - e.g. In this example, an hypothesis is a 2D line from two points.



Outlier Rejection) RANSAC (Random Sample Consensus; 1981)

- RANSAC

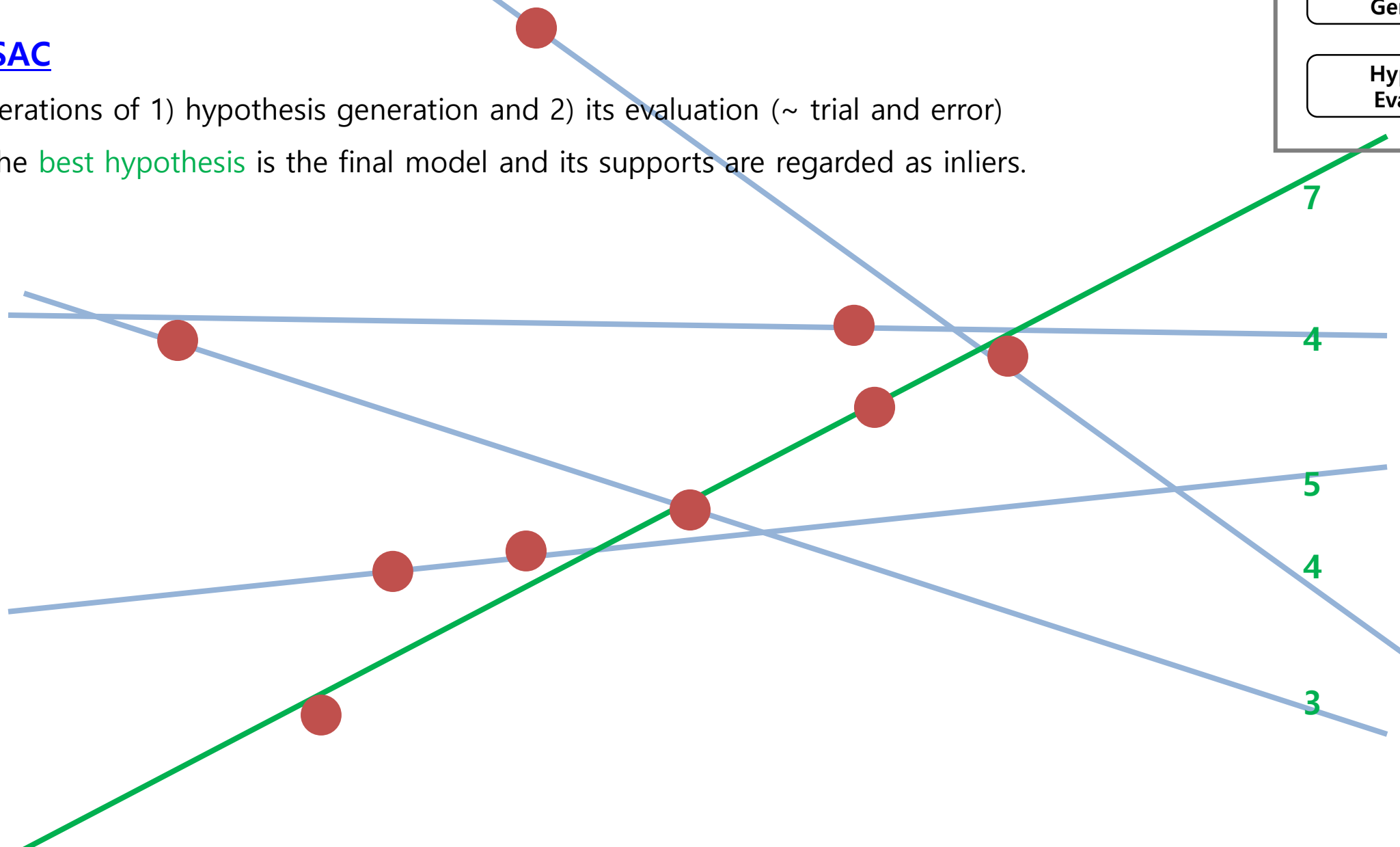
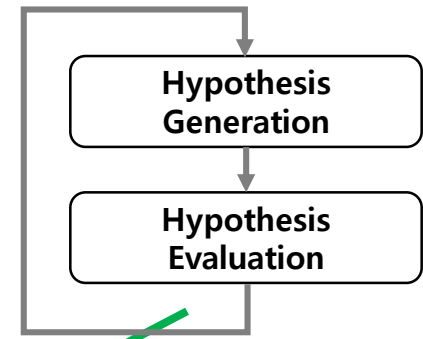
- Iterations of 1) hypothesis generation and 2) its evaluation (~ trial and error)
 - e.g. In this example, the 2D line is supported by points within the given threshold τ .



Outlier Rejection) RANSAC (Random Sample Consensus; 1981)

- RANSAC

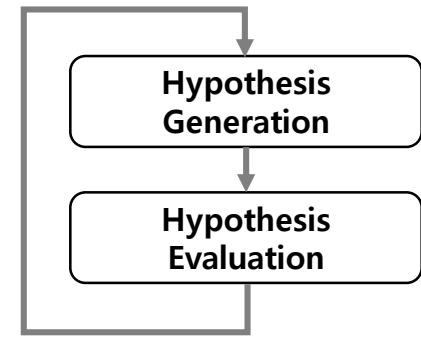
- Iterations of 1) hypothesis generation and 2) its evaluation (~ trial and error)
- The **best hypothesis** is the final model and its supports are regarded as inliers.



Outlier Rejection) RANSAC (Random Sample Consensus; 1981)

▪ RANSAC

- Iterations of 1) hypothesis generation and 2) its evaluation (\sim trial and error)
- How many iterations t are required?
 - Parameters and assumptions
 - s : Success probability (confidence level)
 - d : The number of samples for model generation
 - γ : Inlier ratio
 - Success criteria: **Selecting d samples from inliers within t iterations**
 - Failure sampling: $1 - \gamma^d$
 - Failure probability: $1 - s = (1 - \gamma^d)^t$
 - **The minimum number of iteration for success: $t = \frac{\log(1-s)}{\log(1-\gamma^d)}$**



■ Example) Line fitting with RANSAC [line_fitting_ransac.cpp]

```
1. #include "opencv2/opencv.hpp"

2. // Convert a line format, [n_x, n_y, x_0, y_0] to [a, b, c]
3. // Note) A line model in OpenCV: n_x * (x - x_0) = n_y * (y - y_0)
4. #define CONVERT_LINE(line) (cv::Vec3d(line[0], -line[1], -line[0] * line[2] + line[1] * line[3]))

5. int main()
6. {
7.     cv::Vec3d truth(1.0 / sqrt(2.0), 1.0 / sqrt(2.0), -240.0); // The line model: a*x + b*y + c = 0 (a^2 + b^2 = 1)
8.     int ransac_trial = 50, ransac_n_sample = 2;
9.     double ransac_thresh = 3.0; // 3 x 'data_inlier_noise'
10.    int data_num = 1000;
11.    double data_inlier_ratio = 0.5, data_inlier_noise = 1.0;

12.    // Generate data
13.    std::vector<cv::Point2d> data;
14.    cv::RNG rng;
15.    for (int i = 0; i < data_num; i++)
16.    {
17.        if (rng.uniform(0.0, 1.0) < data_inlier_ratio)
18.        {
19.            double x = rng.uniform(0.0, 480.0);
20.            double y = (truth(0) * x + truth(2)) / -truth(1);
21.            x += rng.gaussian(data_inlier_noise);
22.            y += rng.gaussian(data_inlier_noise);
23.            data.push_back(cv::Point2d(x, y)); // Inlier
24.        }
25.        else data.push_back(cv::Point2d(rng.uniform(0.0, 640.0), rng.uniform(0.0, 480.0))); // Outlier
26.    }

27.    // Estimate a line using RANSAC ...
55.    // Estimate a line using least-squares method (for reference) ...

59.    // Display estimates
60.    printf("* The Truth: %.3f, %.3f, %.3f\n", truth[0], truth[1], truth[2]);
61.    printf("* Estimate (RANSAC): %.3f, %.3f, %.3f (Score: %d)\n", best_line[0], best_line[1], ..., best_score);
62.    printf("* Estimate (LSM): %.3f, %.3f, %.3f\n", lsm_line[0], lsm_line[1], lsm_line[2]);
63.    return 0;
64. }
```

$$t > \frac{\log(1-s)}{\log(1-\gamma^d)} = \frac{\log(1-0.999)}{\log(1-0.5^2)} = 24$$

```

27. // Estimate a line using RANSAC
28. int best_score = -1;
29. cv::Vec3d best_line;
30. for (int i = 0; i < ransac_trial; i++)
31. {
32.     // Step 1: Hypothesis generation
33.     std::vector<cv::Point2d> sample;
34.     for (int j = 1; j < ransac_n_sample; j++)
35.     {
36.         int index = rng.uniform(0, int(data.size()));
37.         sample.push_back(data[index]);
38.     }
39.     cv::Vec4d nnxy;
40.     cv::fitLine(sample, nnxy, CV_DIST_L2, 0, 0.01, 0.01);
41.     cv::Vec3d line = CONVERT_LINE(nnxy);

42.     // Step 2: Hypothesis evaluation
43.     int score = 0;
44.     for (size_t j = 0; j < data.size(); j++)
45.     {
46.         double error = fabs(line(0) * data[j].x + line(1) * data[j].y + line(2));
47.         if (error < ransac_thresh) score++;
48.     }

49.     if (score > best_score)
50.     {
51.         best_score = score;
52.         best_line = line;
53.     }
54. }

55. // Estimate a line using least squares method (for reference)
56. cv::Vec4d nnxy;
57. cv::fitLine(data, nnxy, CV_DIST_L2, 0, 0.01, 0.01);
58. cv::Vec3d lsm_line = CONVERT_LINE(nnxy);

```

Line Fitting Result

- * The Truth: 0.707, 0.707, -240.000
- * Estimate (RANSAC): 0.712, 0.702, -242.170 (Score: 434)
- * Estimate (LSM): 0.748, 0.664, -314.997

Outlier Rejection) Least Squares Method, RANSAC, and M-estimator

▪ Least Squares Method

- Find a model while minimizing sum of squared errors, $\operatorname{argmin}_{\mathbf{m}} \sum_i e(\mathbf{x}_i; \mathbf{m})^2$

▪ RANSAC

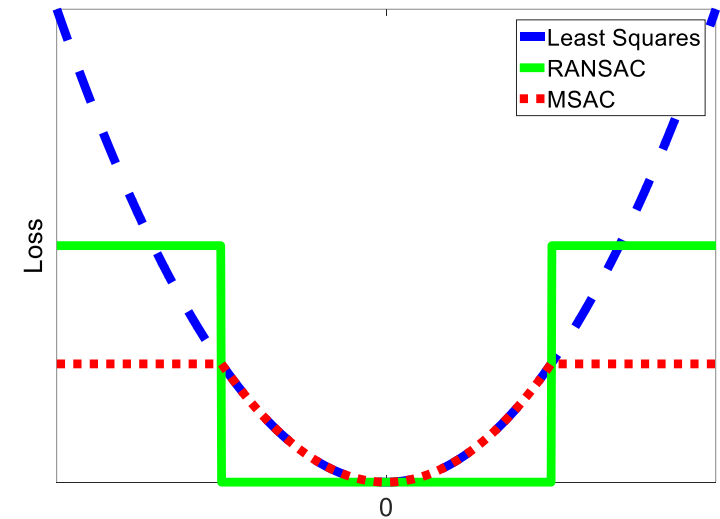
- Find a model while maximizing the number of supports (\sim inlier candidates)
 \sim minimizing the number of outlier candidates

▪ Q) Why RANSAC was robust to outliers?

- Problem formulation with a **loss function** ρ : $\operatorname{argmin}_{\mathbf{m}} \sum_i \rho(e(\mathbf{x}_i; \mathbf{m}))$
- Loss function of least squares method: $\rho(x) = x^2$
- Loss function of RANSAC: $\rho(x) = \begin{cases} 0 & \text{if } |x| < \tau \\ 1 & \text{otherwise} \end{cases}$

▪ M-estimator (\sim weighted least squares) | MSAC (in OpenCV)

- Find a model while minimizing sum of (squared) errors **with a truncated loss function**
- Loss function of M-estimator and MSAC: $\rho(x) = \begin{cases} x^2 & \text{if } |x| < \tau \\ \tau^2 & \text{otherwise} \end{cases}$



One-page Tutorial for Ceres Solver

▪ What is Ceres Solver? [\[Homepage\]](#)

- An **open source C++** library for modelling and solving large and complicated **optimization** problems.
- Problem types: 1) **Non-linear least squares** (with bounds), 2) General unconstrained minimization



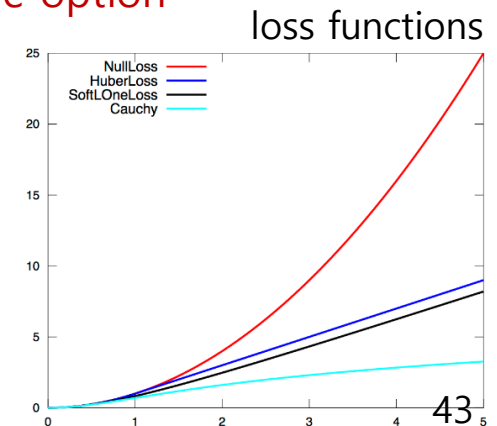
Ceres (an asteroid)

▪ Solving Non-linear Least Squares: $\hat{x} = \underset{x}{\operatorname{argmin}} \sum_i \rho_i(\|r_i(x)\|^2)$

1. Define residual functions (or cost function or error function)
2. Instantiate `ceres::Problem` and add residuals using its member function, `AddResidualBlock()`
 - Instantiate each residual r_i in the form of `ceres::CostFunction` and add it
 - Select how to calculate its derivative (Jacobian)
(`ceres::AutoDiffCostFunction` or `ceres::NumericDiffCostFunction` or `ceres::ScaledCostFunction`)
 - Instantiate its `ceres::LossFunction` ρ_i and add it (if the problem needs robustness against outliers)
3. Instantiate `ceres::Solver::Options` (and also `ceres::Solver::Summary`) and configure the option
4. Run `ceres::Solve()`

▪ Solving General Minimization: $\hat{x} = \underset{x}{\operatorname{argmin}} f(x)$

- `ceres::CostFunction` → `ceres::FirstOrderFunction`, `ceres::GradientFunction`
- `ceres::Problem` → `ceres::GradientProblem`
- `ceres::Solver` → `ceres::GradientProblemSolver`



■ Example: Line fitting with M-estimator [line_fitting_m_estimator.cpp]

```

1. #include "opencv2/opencv.hpp"
2. #include "ceres/ceres.h"
3. ...
4. struct GeometricError
5. {
6.     GeometricError(const cv::Point2d& pt) : datum(pt) { }
7.     template<typename T>
8.     bool operator()(const T* const line, T* residual) const
9.     {
10.         residual[0] = (line[0] * T(datum.x) + line[1] * T(datum.y) + line[2]) / sqrt(line[0] * line[0] + line[1] * line[1]);
11.         return true;
12.     }
13. private:
14.     const cv::Point2d datum;
15. };

16. int main()
17. {
18.     ...
19.     // Estimate a line using M-estimator
20.     cv::Vec3d opt_line(1, 0, 0);
21.     ceres::Problem problem;
22.     for (size_t i = 0; i < data.size(); i++)
23.     {
24.         ceres::CostFunction* cost_func = new ceres::AutoDiffCostFunction<GeometricError, 1, 3>(new GeometricError(data[i]));
25.         ceres::LossFunction* loss_func = NULL;
26.         if (loss_width > 0) loss_func = new ceres::CauchyLoss(loss_width);
27.         problem.AddResidualBlock(cost_func, loss_func, opt_line.val);
28.     }
29.     ceres::Solver::Options options;
30.     options.linear_solver_type = ceres::ITERATIVE_SCHUR;
31.     options.num_threads = 8;
32.     options.minimizer_progress_to_stdout = true;
33.     ceres::Solver::Summary summary;
34.     ceres::Solve(options, &problem, &summary);
35.     std::cout << summary.FullReport() << std::endl;
36.     opt_line /= sqrt(opt_line[0] * opt_line[0] + opt_line[1] * opt_line[1]); // Normalize
37.     ...
38.     return 0;
39. }

```

1) Define a residual as C++ generic function (T ~ double)

Note) The generic is necessary for automatic differentiation.

$$\text{c.f. } e(x, y; a, b, c) = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

2) Instantiate a problem and add a residual for each datum

The dimension of a residual

The dimension of the first model parameter

3) Instantiate options and configure it

4) Solve the minimization problem

Summary) Outlier Rejection (Robust Parameter Estimation)

- **Bottom-up Approaches (~ Voting)**

- ~~Hough transform~~

- A single datum votes multiple model parameter candidates.
 - Note) The parameter space is maintained as a multi-dimensional histogram (discretization).
 - Score: The number of hits by data

- **RANSAC family**

- A sample of data votes a single model parameter candidate.
 - Score: The number of supports (whose error is within threshold)
 - Note) Application examples: Line fitting, homography estimation, relative pose estimation, PnP

- **Top-down Approaches**

- **Optimization with truncated loss functions (e.g. M-estimator)**

- The initial model parameter moves along with the gradient of the given cost function with whole data.
 - Score: A cost function
 - Note) The cost function includes a truncated loss function.
 - Note) Application examples: Camera calibration, visual SLAM, SfM

Summary

- **Feature Points**

- Gradient-based: Harris corner, GFTT corner, SIFT, SURF
- Intensity-based: FAST
- DL-based: LIFT, SuperPoint

- **Feature Descriptors**

- Real-valued: SIFT, SURF (DL-based: LIFT, SuperPoint)
- Binary-valued: BRIEF, ORB

- **Feature Matching**

- Distance measures
- Matching methods

- **Feature Tracking**

- LK optical flow → KLT tracker

- **Outlier Rejection**

- RANSAC → MSAC
- Least squares method → M-estimation

