Slides and example codes are available:
https://github.com/mint-lab/cv_tutorial

Visual Features

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

Getting Started from a Quiz

600 pixels

What is it?

Getting Started from a Quiz

What is it?

600 pixels

Getting Started from a Quiz

- **Why corners (junction)?**
  - a.k.a. keypoints, interest points, salient points, and feature points
  - Note) ⊂ local invariant features (e.g. corner, edge, region, …)

"Duck"

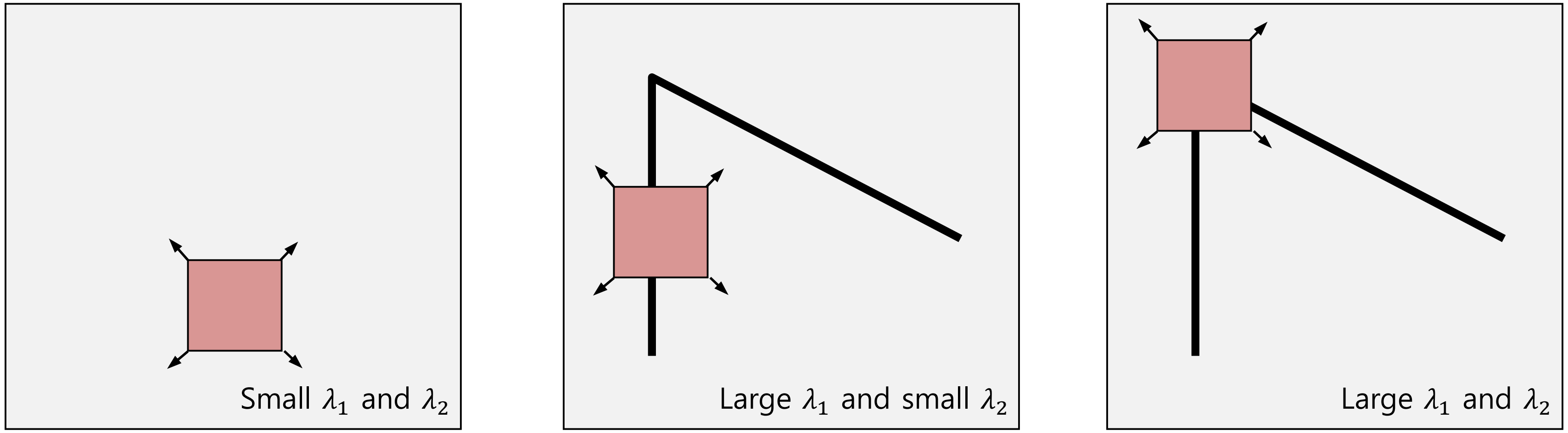
600 pixels | 1095 pixels | 600 pixels

- **Requirements of local invariance features**
  - **repeatability** (invariance, robustness), **distinctiveness**, **locality** (due to occlusion)
  - quantity, accuracy, efficiency

[Reference] Tuytelaars et al., **Local Invariant Feature Detectors: A Survey**, Foundations and Trends in Computer Graphics and Vision, 2008

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

$$C(\Delta_x, \Delta_y) = \sum_{(x,y)\in W} \left(I(x+\Delta_x, y+\Delta_y) - I(x,y) \right)^2$$

$$\approx \begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \underbrace{\begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix}}_{M} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$$

$$\text{c.f. } I(x+\Delta_x, y+\Delta_y) \approx I(x,y) + \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \text{ and } I_x = \frac{\partial I}{\partial x}$$

Harris corner response:

$$R = \det(M) - k\,\mathrm{trace}(M)^2$$

$$\text{c.f. } \det(M) = \lambda_1 \lambda_2\,,\ \mathrm{trace}(M) = \lambda_1 + \lambda_2\,,\ k \in [0.04, 0.06]$$

Note) Good-Feature-to-Track (Shi-Tomasi; 1994):

$$R = \min(\lambda_1, \lambda_2)$$

Image: *Matching with Invariant Features* (by Frolova and Simakov, Lecture Slides, 2004).

Harris Corner (1988)

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

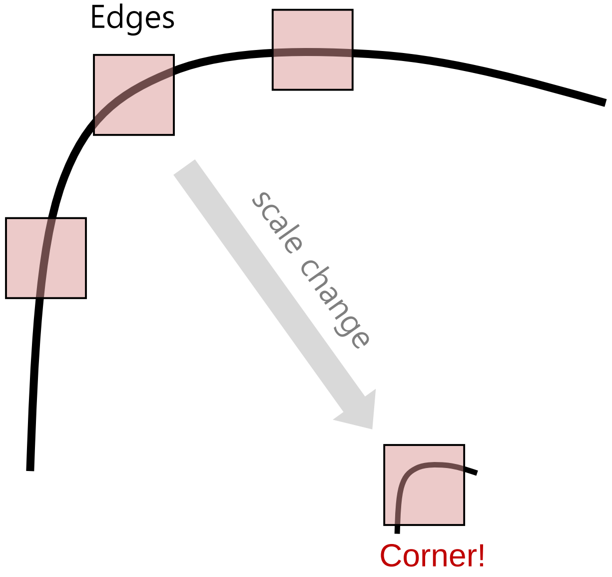
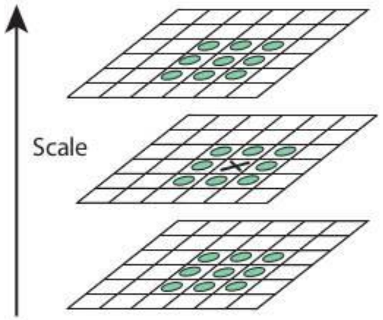


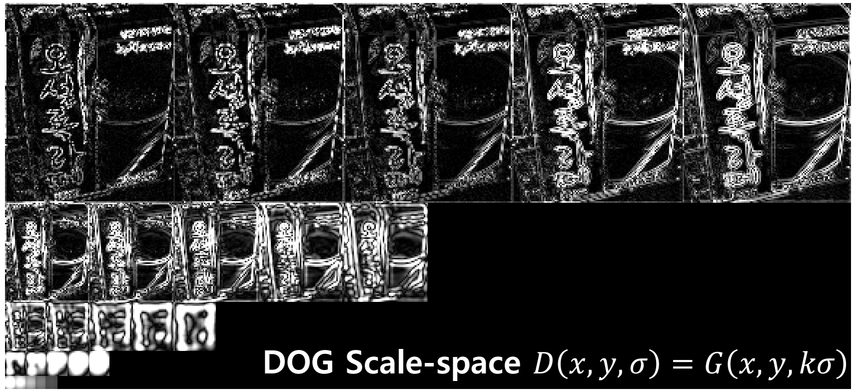
Image: *Matching with Invariant Features* (by Frolova and Simakov, Lecture Slides, 2004).

SIFT (Scale-Invariant Feature Transform; 1999)

- Key idea: **Scale-space** (~ image pyramid)

Gaussian Scale-space $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$





DOG Scale-space $D(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma)$

- Part #1) **Feature point detection**
  1. Find **local extrema** (minima and maxima) in DOG scale-space
  2. Localize its position accurately (sub-pixel level) using 3D quadratic function
  3. Eliminate **low contrast candidates**, $|D(\mathbf{x})| < \tau$
  4. Eliminate **candidates on edges**, $\frac{\mathrm{trace}(H)^2}{\det(H)} < \frac{(r+1)^2}{r}$ where $H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$

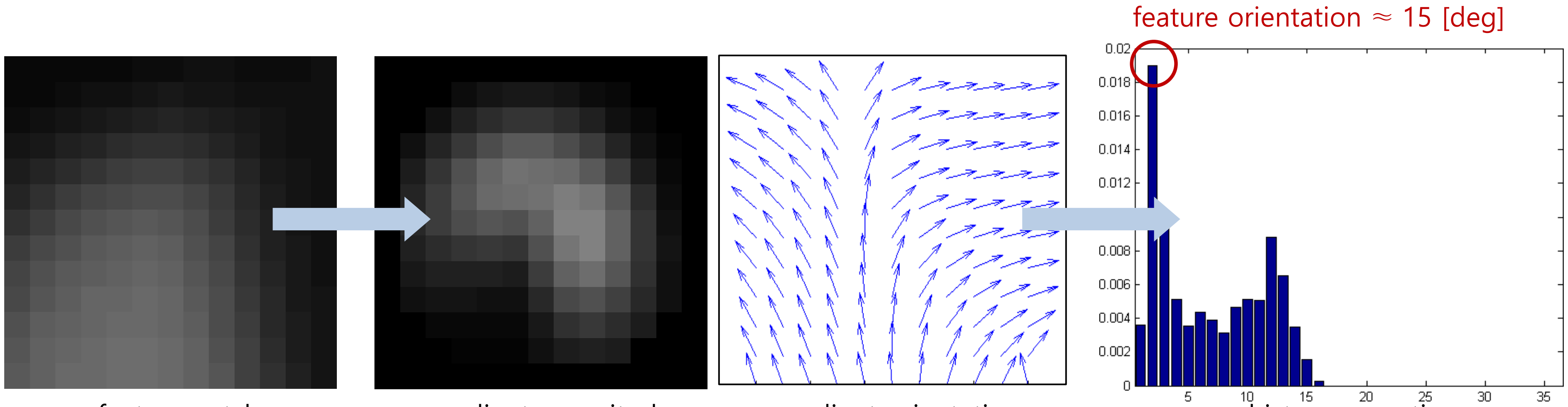
SIFT (Scale-Invariant Feature Transform; 1999)

- Part #2) **Orientation assignment**
  1. Derive magnitude and orientation of gradient of each patch

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

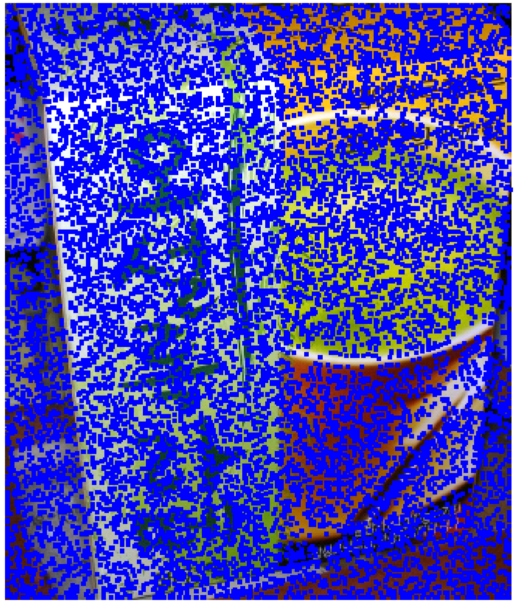
$$\theta(x,y) = \tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$$

  2. Find the strongest orientation
     - Histogram voting (36 bins) with Gaussian-weighted magnitude



feature patch | gradient magnitude | gradient orientation | histogram voting

SIFT (Scale-Invariant Feature Transform; 1999)



local extrema (N: 11479)

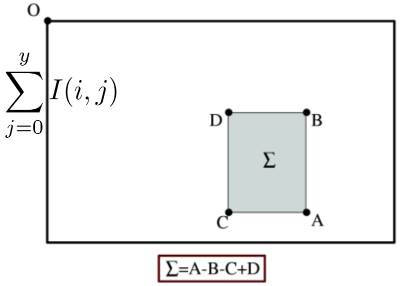
feature points (N: 971)

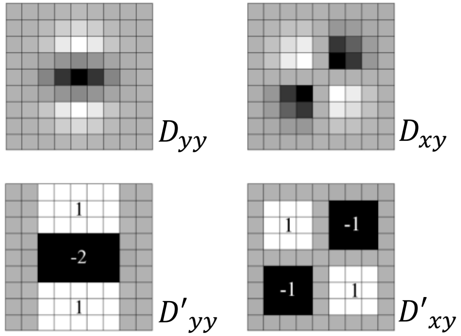
feature scales and orientations

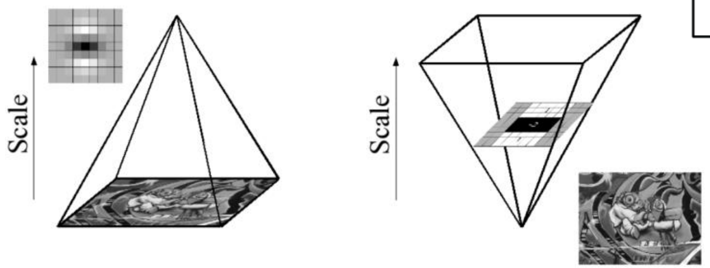
Note) SURF (Speeded Up Robust Features; 2006):

- Key idea: Approximation of SIFT using **integral image** and …

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



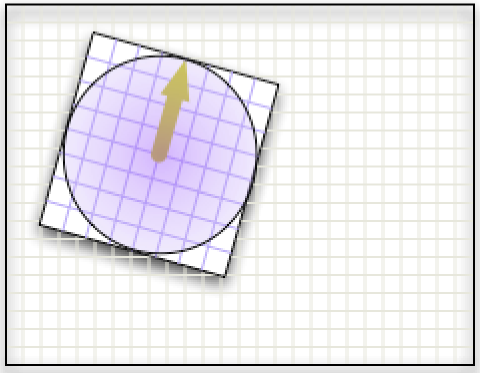


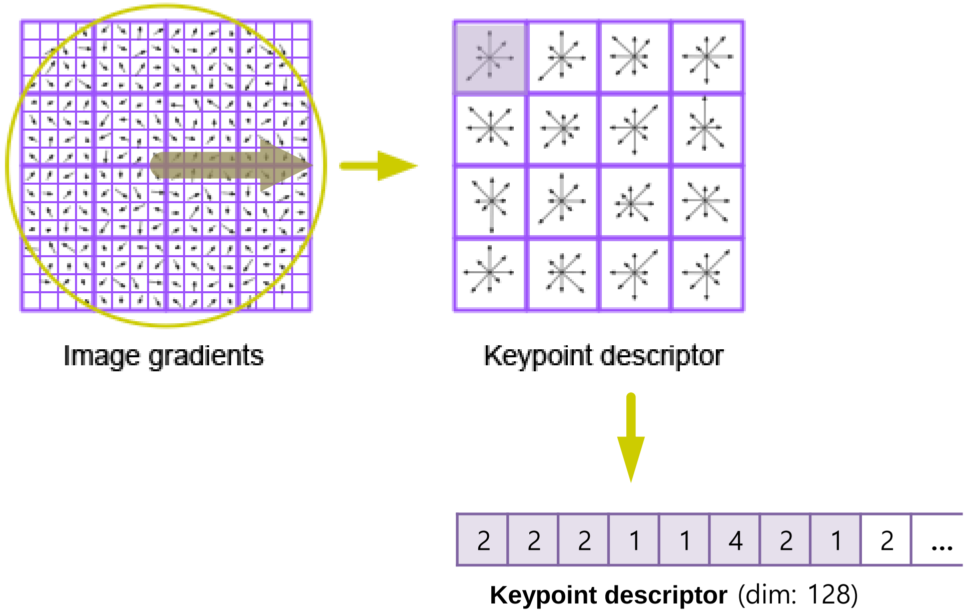


[Reference] Bay et. al., **SURF: Speeded Up Robust Features**, CVIU, 2008

SIFT (Scale-Invariant Feature Transform; 1999)

- Part #3) **Feature descriptor extraction**
  - 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
  - Encode the histogram into a 128-dimensional vector

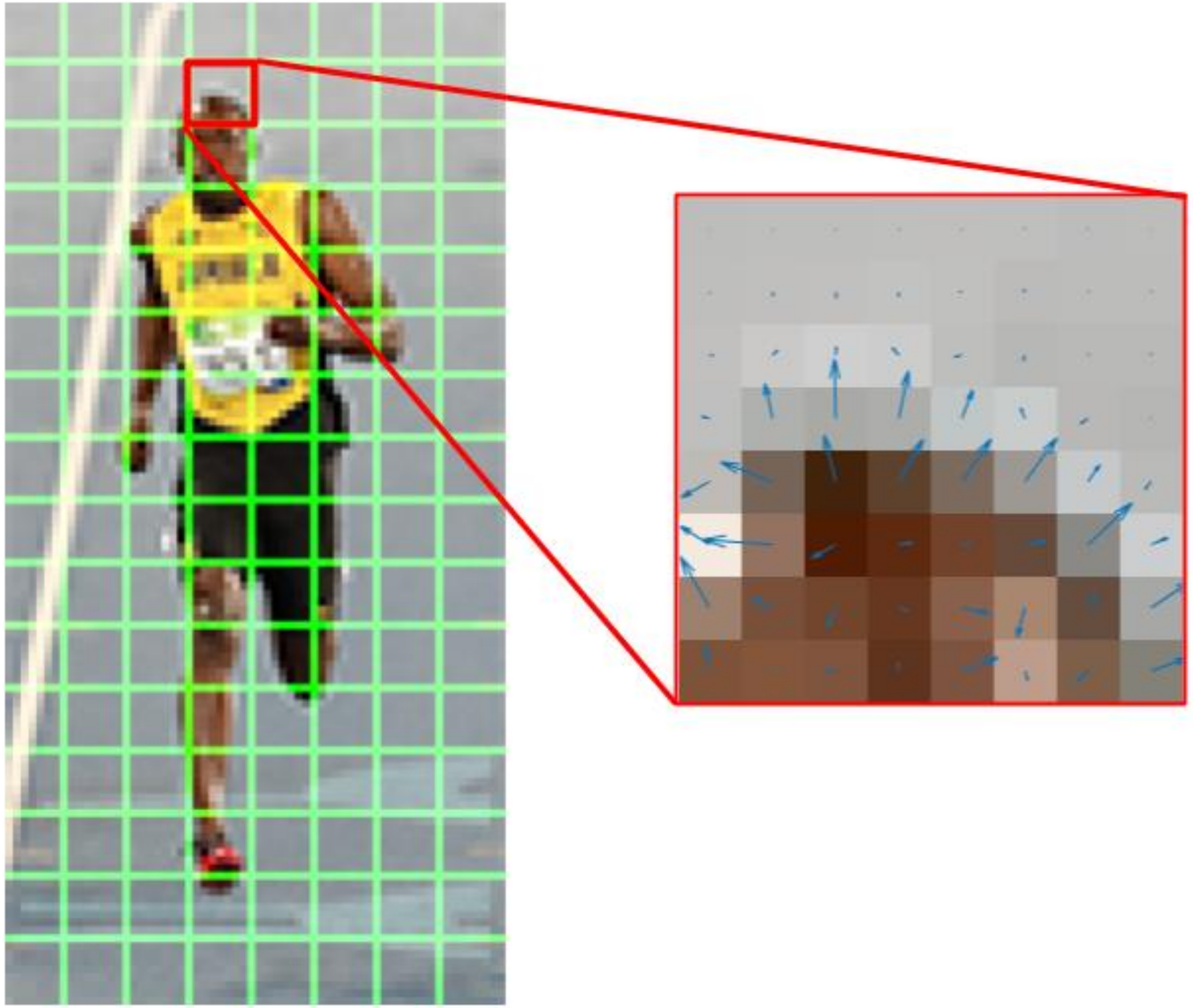




feature scales and orientations

Image: Jonas Hurrelmann (https://www.eecs.tu-berlin.de/fileadmin/fg144/Courses/06WS/scanning/Jonas/html/index.html).

HOG (Histogram of Oriented Gradients)



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

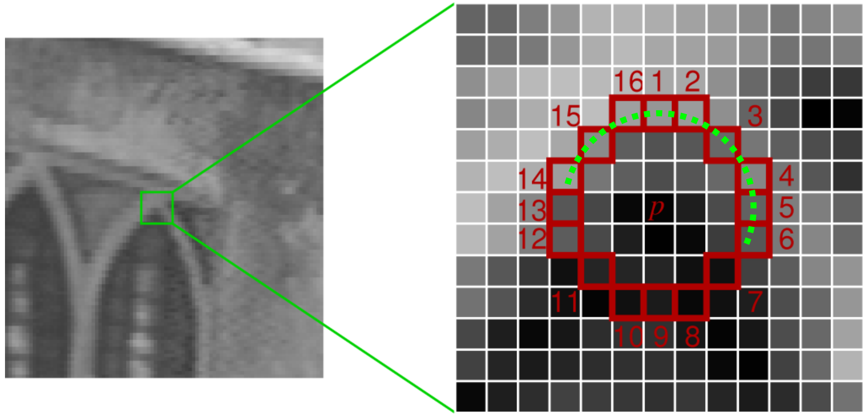
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

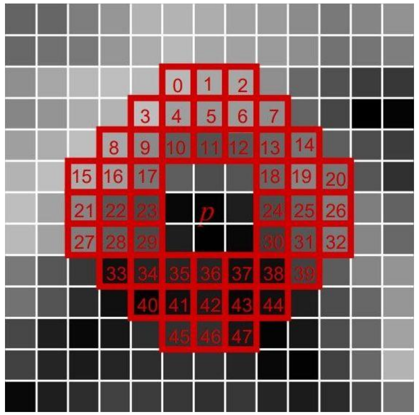
Image: [LearnOpenCV]

FAST (Features from Accelerated Segment Test; 2006)

- Key idea: **Continuous arc of N or more pixels**
  - Is this patch a corner?
    - Is the segment brighter than $p + t$? Is the segment darker than $p - t$?
    - t: The threshold of similar intensity
  - Too many corners! it needs non-maximum suppression.



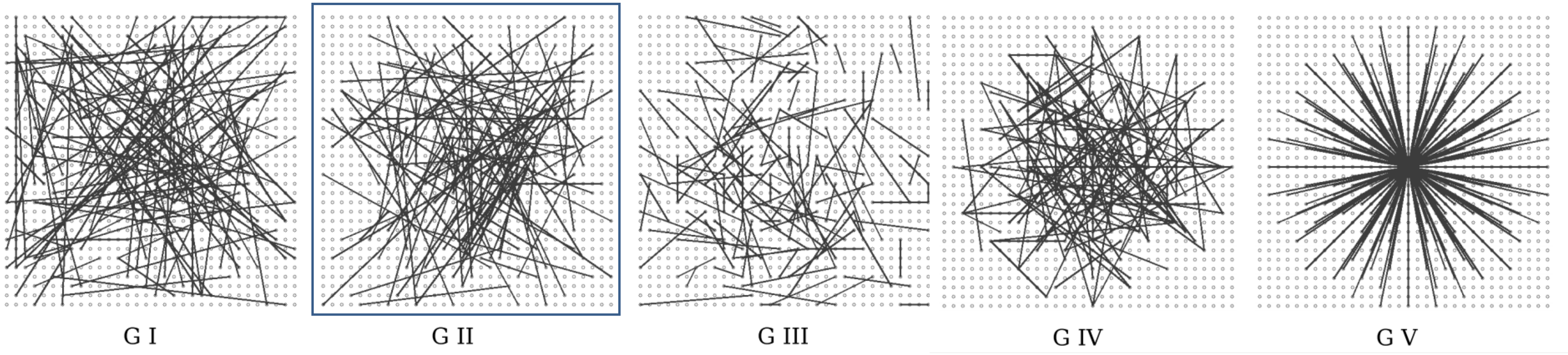
- Versions
  - **FAST-9** (N: 9), FAST-12 (N: 12), …
  - FAST-**ER**: Training a decision tree to <u>**e**</u>nhance <u>**r**</u>epeatability with more pixels



[Reference] Rosten et al., **FASTER and better: A machine learning approach to corner detection**, T-PAMI, 2010

BRIEF (Binary Robust Independent Elementary Features; 2010)

- Key idea: **A sequence of intensity comparison of random pairs**
  - Applying smoothing for stability and repeatability
  - Path size: 31 x 31 pixels



- Versions: The number of tests
  - BRIEF-32, BRIEF-64, BRIEF-128, BRIEF-256 …
- Examples of combinations
  - CenSurE detector (a.k.a. Star detector) + BRIEF descriptor
  - SURF detector + BRIEF descriptor

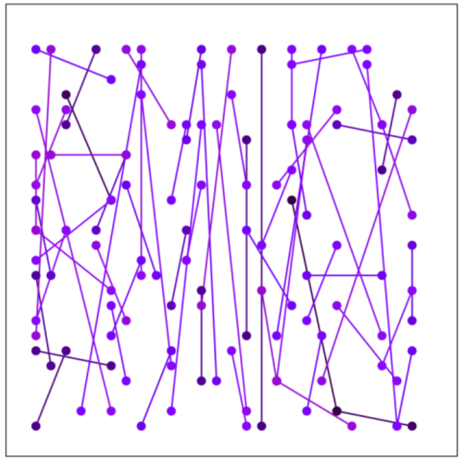
[Reference] Calonder et. al., **BRIEF: Computing a Local Binary Descriptor Very Fast**, T-PAMI, 2012

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*

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

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



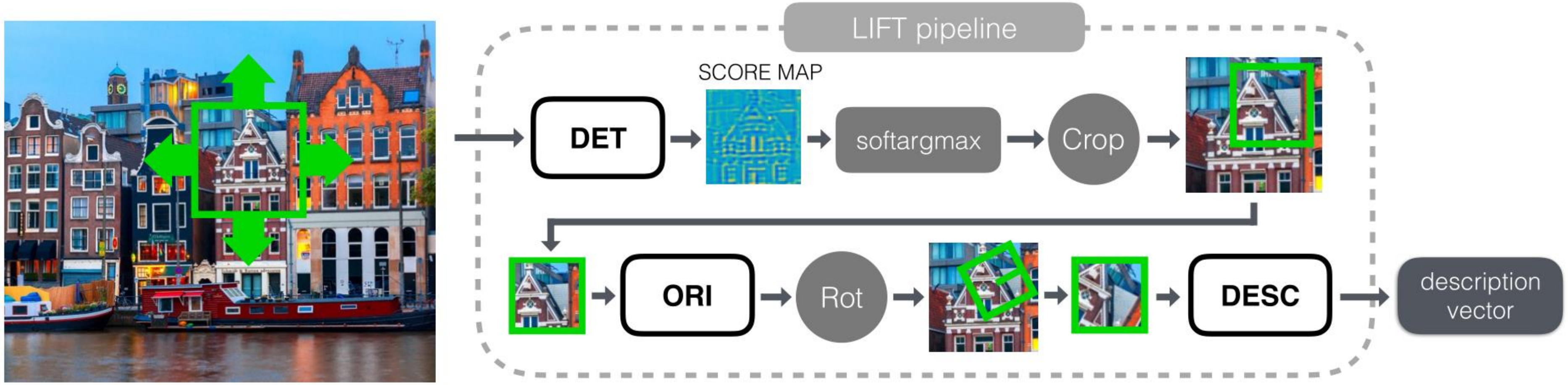
- Combination: **ORB**
  - FAST-9 detector (with orientation) + BRIEF-256 descriptor (with trained pairs)

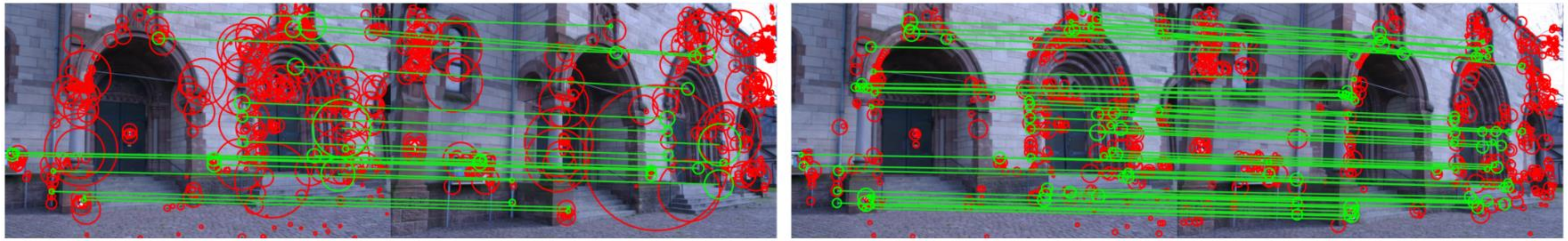
- Computing time
  - ORB: **15.3 [msec]** / SURF: 217.3 [msec] / SIFT: 5228.7 [msec] @ 24 images (640x480) in Pascal dataset

[Reference] Rublee et. al., **ORB: An Efficient Alternative to SIFT or SURF**, ICCV, 2011

LIFT (Learned Invariant Feature Transform; 2016)

- Key idea: **Deep neural network**
  - **DET** (feature detector) + **ORI** (orientation estimator) + **DESC** (feature descriptor)





SIFT LIFT

[Reference] Yi et. al., **LIFT: Learned Invariant Feature Transform**, ECCV, 2016

Lukas-Kanade Optical Flow (1981)

- Key idea: **Finding movement of a patch**

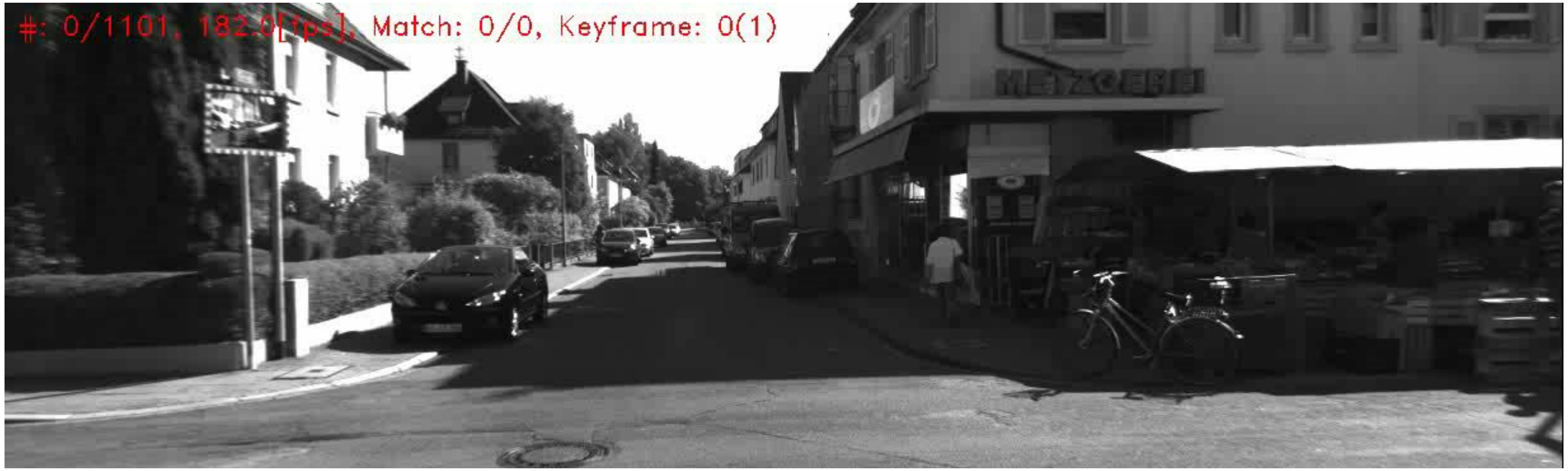
  Brightness constancy constraint: $I(x, y, t) = I\left(x + \Delta_x, y + \Delta_y, t + \Delta_t\right)$
  (if same patch)

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

  $$\mathrm{A}\boldsymbol{v} = \boldsymbol{b}$$ where $\mathrm{A} = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}$, $\boldsymbol{v} = \begin{bmatrix} V_x \\ V_y \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} -I_t(p_1) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$, and $p_i \in W$

  $$\therefore \boldsymbol{v} = \mathrm{A}^{\dagger} \boldsymbol{b} = (\mathrm{A}^{\intercal}\mathrm{A})^{-1}\mathrm{A}^{\intercal} \boldsymbol{b}$$

- Combination: **KLT tracker**
  - Shi-Tomasi detector (a.k.a. GFTT) + Lukas-Kanade optical flow

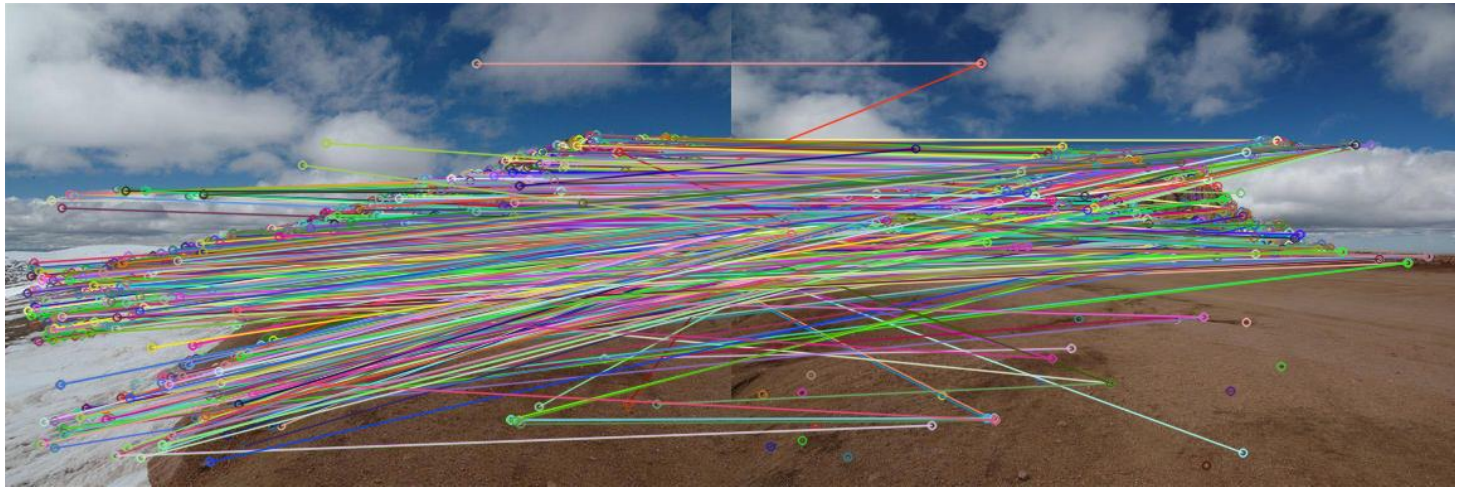


Overview of Feature Correspondence

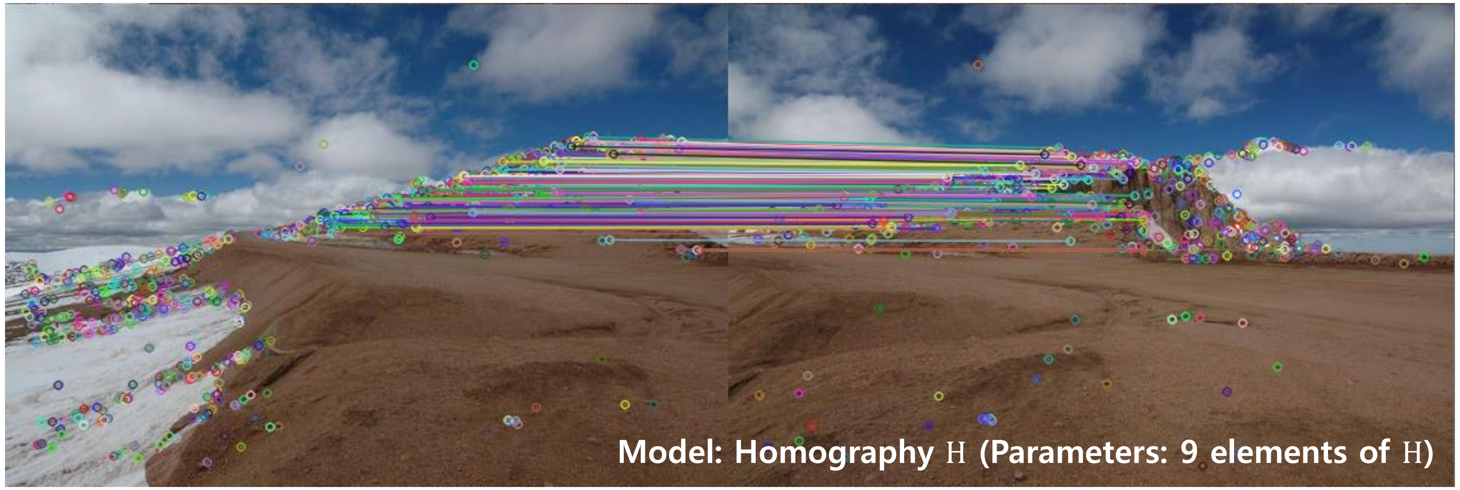
- **Features**
  - **Corners**: Harris corner, GFTT (Shi-Tomasi corner), SIFT, SURF, FAST, LIFT, …
  - Edges, line segments, regions, …
- **Feature Descriptors and Matching**
  - **Patch**: Raw intensity
    - Measures: SSD (sum of squared difference), ZNCC (zero normalized cross correlation), …
  - **Floating-point descriptors**: SIFT, SURF, (DAISY), LIFT, … → e.g. A 128-dim. vector (a histogram of gradients)
    - Measures: Euclidean distance, cosine distance, (the ratio of first and second bests)
    - Matching: Brute-force matching ($O(N^2)$), ANN (approximated nearest neighborhood) search ($O(\mathrm{log}N)$)
    - Pros (+): **High discrimination power**
    - Cons (–): **Heavy computation**
  - **Binary descriptors**: BRIEF, ORB, (BRISK), (FREAK), … → e.g. A 128-bit string (a series of intensity comparison)
    - Measures: Hamming distance
    - Matching: Brute-force matching ($O(N^2)$)
    - Pros (+): **Less storage and faster extraction/matching**
    - Cons (–): **Less performance**
- **Feature Tracking (a.k.a. Optical Flow)**
  - **Optical flow**: (Horn-Schunck method), Lukas-Kanade method
    - Measures: SSD (sum of squared difference)
    - Tracking: Finding displacement of a similar patch
    - Pros (+): **No descriptor and matching** (faster and compact)
    - Cons (–): **Not working in wide baseline**

Why Outliers?

Putative matches (inliers + outliers)



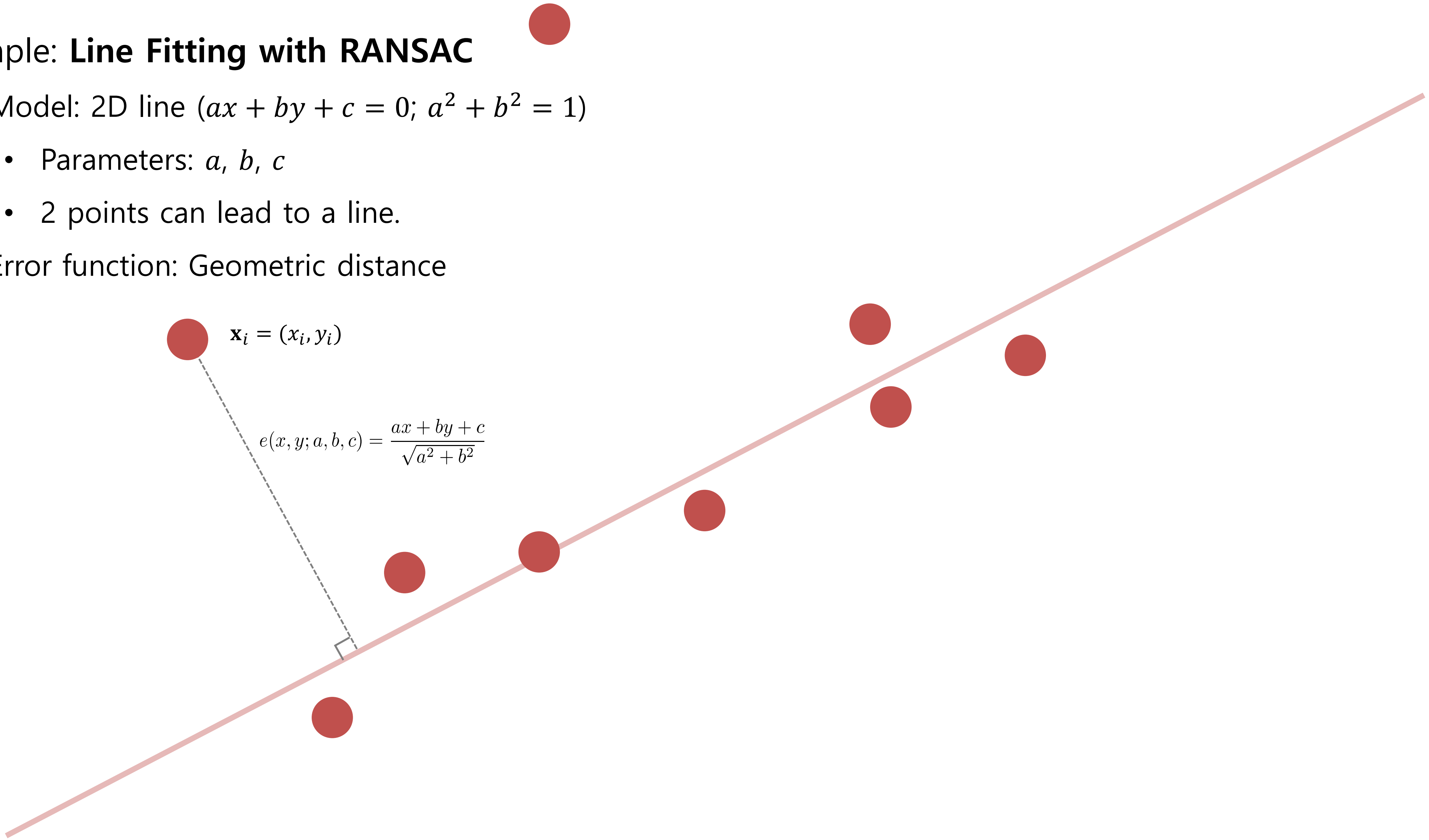
After applying RANSAC (inliers)



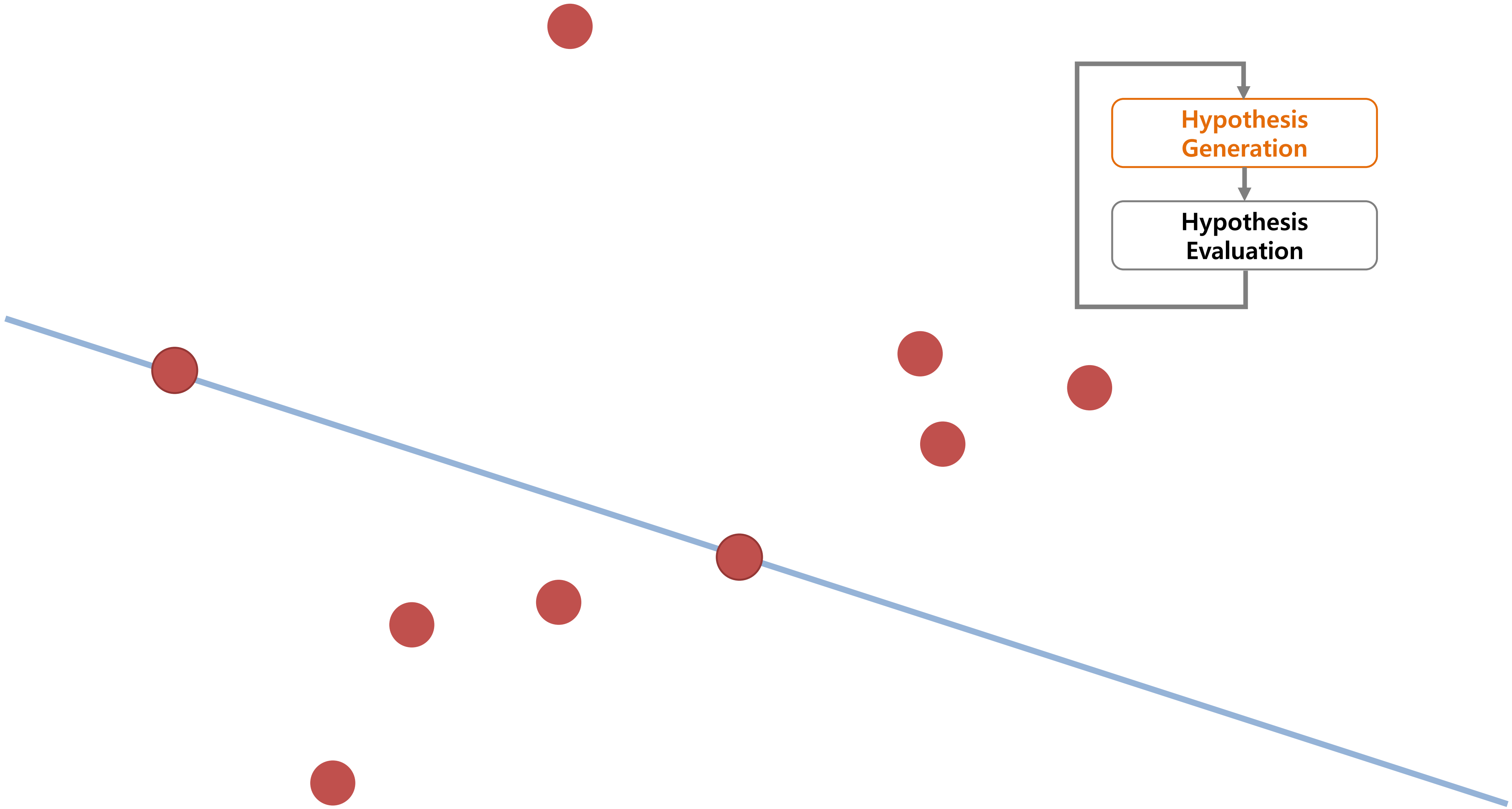
Model: Homography H (Parameters: 9 elements of H)

RANSAC: Random Sample Consensus

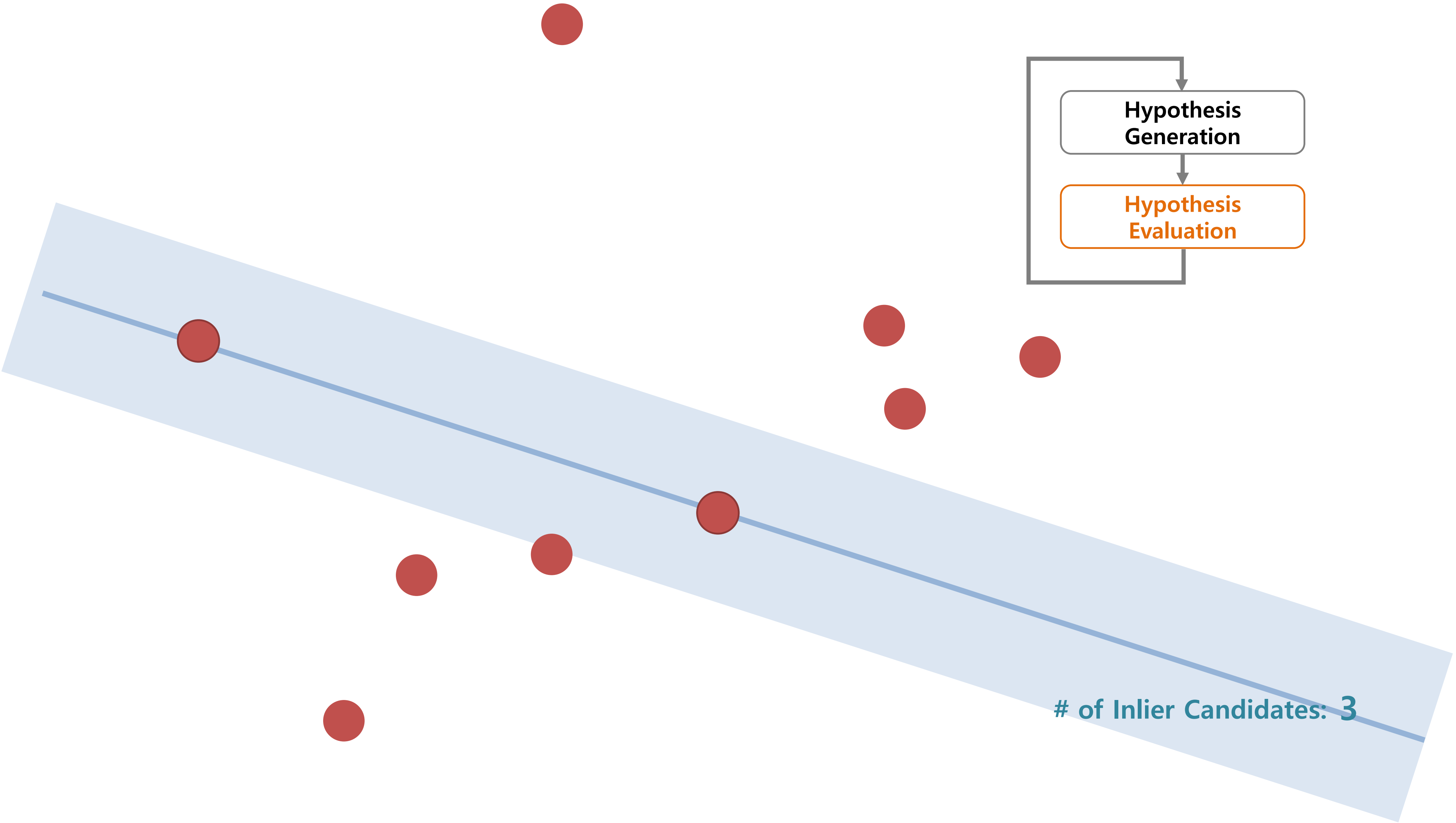
- Example: **Line Fitting with RANSAC**
  - Model: 2D line ($ax + by + c = 0$; $a^2 + b^2 = 1$)
    - Parameters: a, b, c
    - 2 points can lead to a line.
  - Error function: Geometric distance



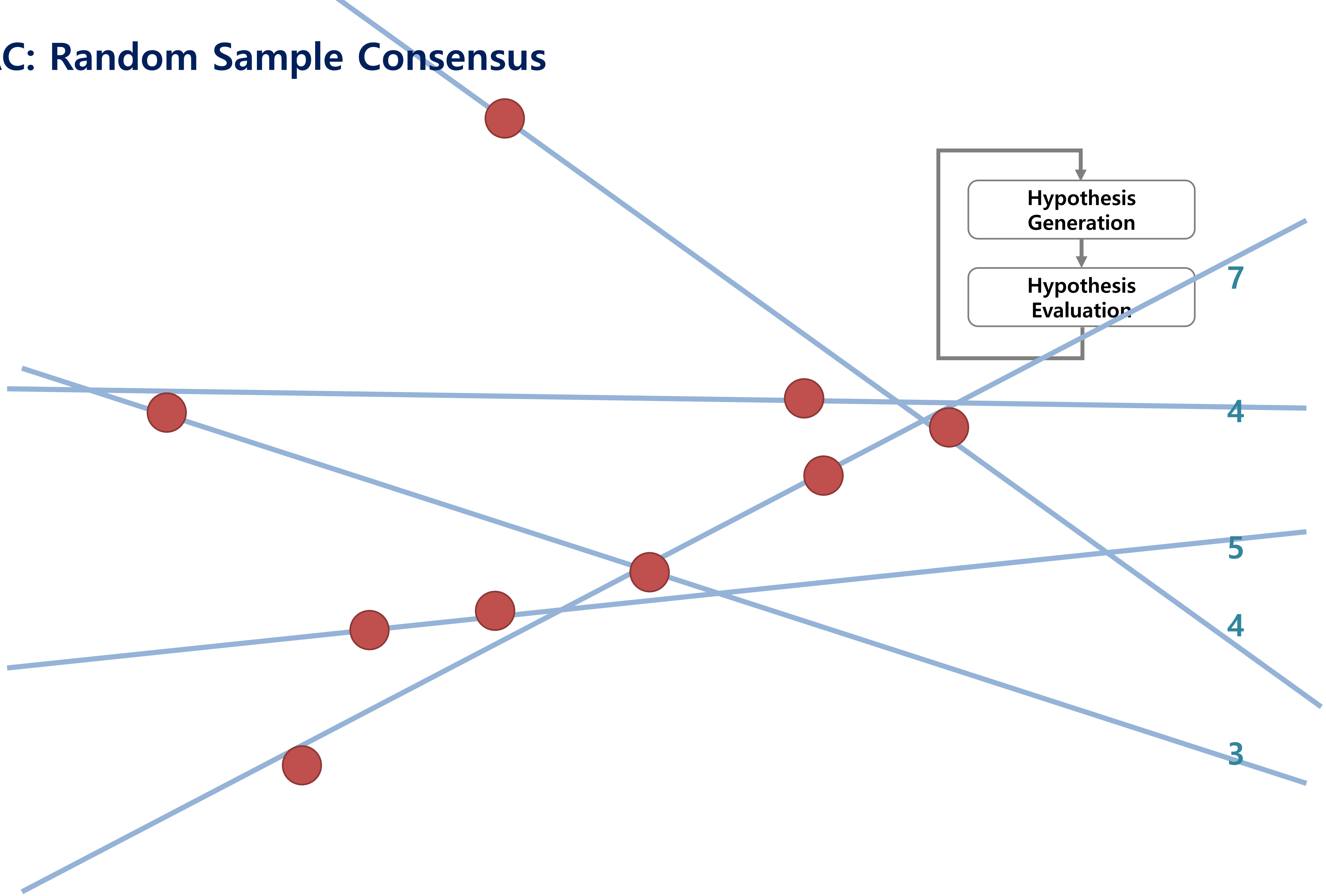
RANSAC: Random Sample Consensus



RANSAC: Random Sample Consensus



RANSAC: Random Sample Consensus

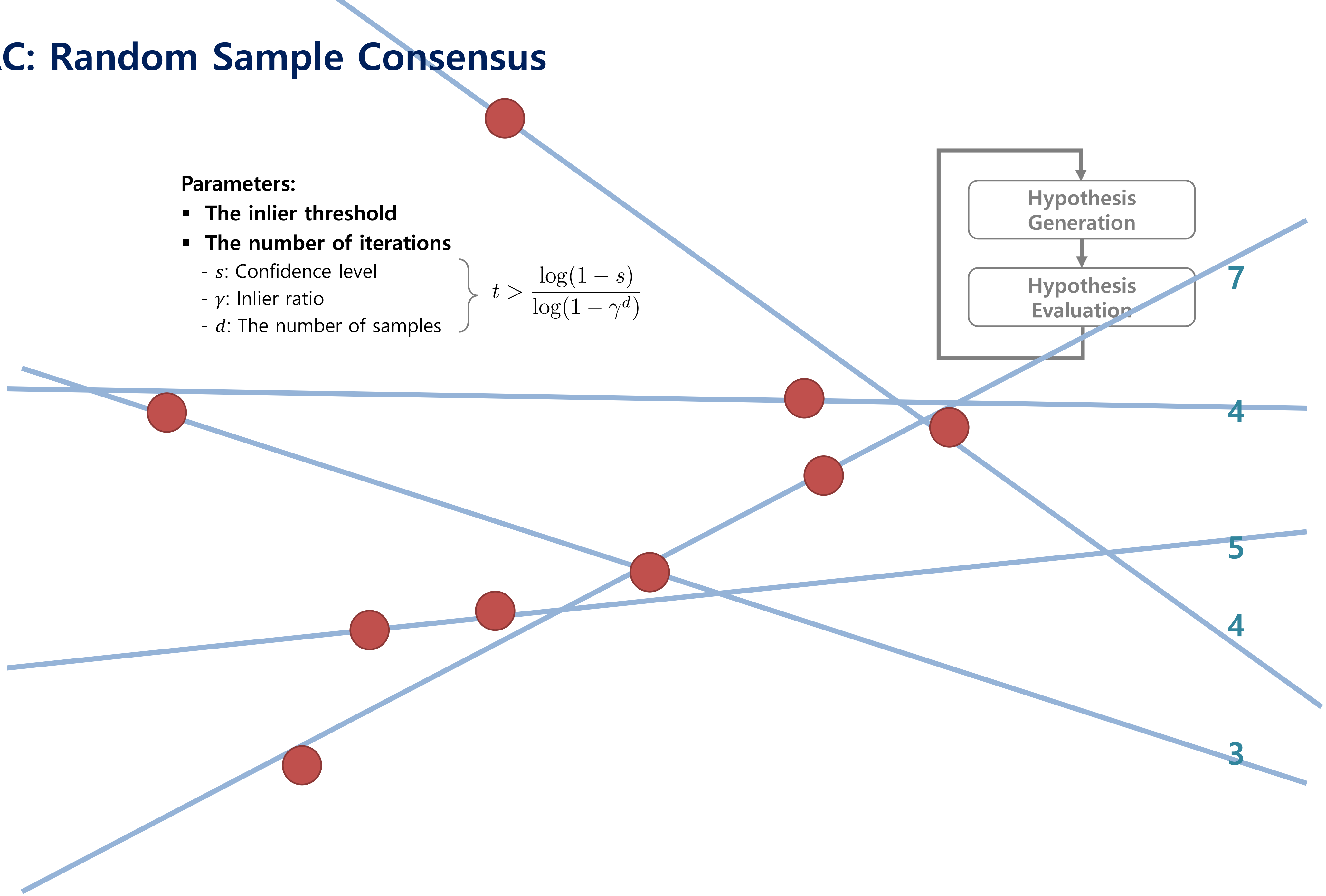


RANSAC: Random Sample Consensus

Parameters:

- **The inlier threshold**
- **The number of iterations**
  - s: Confidence level
  - γ: Inlier ratio
  - d: The number of samples

$$t > \frac{\log(1-s)}{\log(1-\gamma^d)}$$



- Example: **Line Fitting with RANSAC** [line_fitting_ransac.cpp]

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

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

int main()
{
    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)
    int ransac_trial = 50, ransac_n_sample = 2;
    double ransac_thresh = 3.0; // 3 x 'data_inlier_noise'
    int data_num = 1000;
    double data_inlier_ratio = 0.5, data_inlier_noise = 1.0;

    // Generate data
    std::vector<cv::Point2d> data;
    cv::RNG rng;
    for (int i = 0; i < data_num; i++)
    {
        if (rng.uniform(0.0, 1.0) < data_inlier_ratio)
        {
            double x = rng.uniform(0.0, 480.0);
            double y = (truth(0) * x + truth(2)) / -truth(1);
            x += rng.gaussian(data_inlier_noise);
            y += rng.gaussian(data_inlier_noise);
            data.push_back(cv::Point2d(x, y)); // Inlier
        }
        else data.push_back(cv::Point2d(rng.uniform(0.0, 640.0), rng.uniform(0.0, 480.0))); // Outlier
    }

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

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

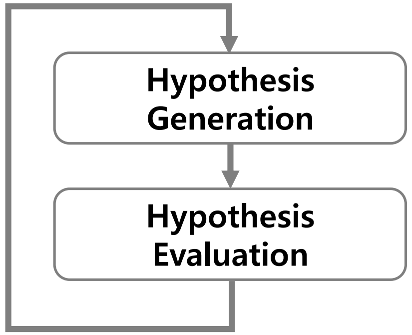
$$\text{c.f.}\quad t > \frac{\log(1-s)}{\log(1-\gamma^d)} = \frac{\log(1-0.999)}{\log(1-0.5^2)} = 24$$

```cpp
// Estimate a line using RANSAC
int best_score = -1;
cv::Vec3d best_line;
for (int i = 0; i < ransac_trial; i++)
{
    // Step 1: Hypothesis generation
    std::vector<cv::Point2d> sample;
    for (int j = 1; j < ransac_n_sample; j++)
    {
        int index = rng.uniform(0, int(data.size()));
        sample.push_back(data[index]);
    }
    cv::Vec4d nnxy;
    cv::fitLine(sample, nnxy, CV_DIST_L2, 0, 0.01, 0.01);
    cv::Vec3d line = CONVERT_LINE(nnxy);

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

    if (score > best_score)
    {
        best_score = score;
        best_line = line;
    }
}

// Estimate a line using least squares method (for reference)
cv::Vec4d nnxy;
cv::fitLine(data, nnxy, CV_DIST_L2, 0, 0.01, 0.01);
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
```

Least Squares Method, RANSAC, and M-estimator

- **Least Squares Method**
  - Find a model while minimizing sum of squared errors, $\arg\min_{a,b,c} \sum_i e(\mathbf{x}_i; a, b, c)^2$

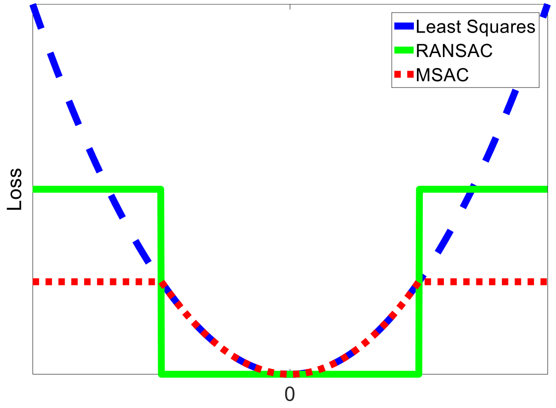
- **RANSAC**
  - Find a model while maximizing the number of supports (~ inlier candidates)

    ~ minimizing the number of outlier candidates

- **Why RANSAC was robust to outliers?**

  $$\text{Problem:} \quad \arg\min_{a,b,c} \sum_i \rho\big(e(\mathbf{x}_i; a, b, c)\big)$$

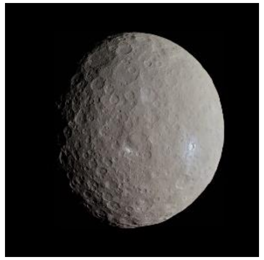
  In the view of **loss functions** ρ,
  - Least squares method: $\rho(x) = x^2$
  - RANSAC: $\rho(x) = \begin{cases} 0 & \text{if } |x| < \tau \\ 1 & \text{o.w.} \end{cases}$



- **M-estimator** (~ weighted least squares) **| MSAC** (in OpenCV)
  - Find a model while minimizing sum of (squared) errors **with a truncated loss function**

[Reference] Choi et al., **Performance Evaluation of RANSAC Family**, BMVC, 2009

One-page Tutorial for Ceres Solver



Ceres (an asteroid)

- **Ceres Solver?**
  - An **open source C++** library for modelling and solving large and complicated **optimization** problems.
    - Since 2010 by Google (BSD license)
  - Problem types: 1) **Non-linear least squares** (with bounds), 2) General unconstrained minimization
  - Homepage: http://ceres-solver.org/

- **Solving Non-linear Least Squares**
  1. Define residual functions (or cost function or error function) $\arg\min_{\mathbf{m}} \sum_i \rho_i\left(\|r_i(\mathbf{m})\|^2 \right)$
  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::SizedCostFunction`)
       - Note) Automatic derivation (using the chain rule) is recommended for convenience and performance.
     - 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** $\arg\min_{\mathbf{x}} f(\mathbf{x})$
  - `ceres::CostFunction` → `ceres::FirstOrderFunction`, `ceres::GradientFunction`
  - `ceres::Problem` → `ceres::GradientProblem`
  - `ceres::Solver` → `ceres::GradientProblemSolver`

loss functions

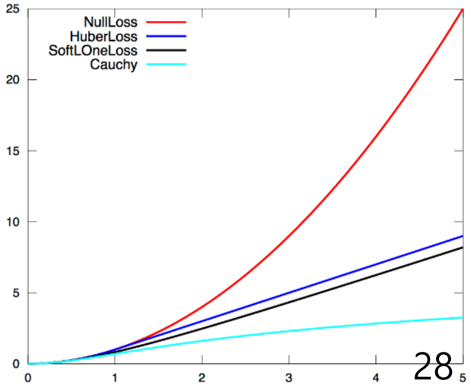


Image: Wikipedia (https://en.wikipedia.org/wiki/Ceres_(dwarf_planet)) and Ceres Solver

Example: **Line Fitting with M-estimator** [line_fitting_m_est.cpp]

```cpp
#include "opencv2/opencv.hpp"
#include "ceres/ceres.h"
...
struct GeometricError
{
    GeometricError(const cv::Point2d& pt) : datum(pt) { }
    template<typename T>
    bool operator()(const T* const line, T* residual) const
    {
        residual[0] = (line[0] * T(datum.x) + line[1] * T(datum.y) + line[2]) / sqrt(line[0] * line[0] + line[1] * line[1]);
        return true;
    }
private:
    const cv::Point2d datum;
};
```

1) Define a residual as C++ generic function (T ~ double)
Note) The generic is necessary for automatic differentiation.

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

```cpp
int main()
{
    ...
    // Estimate a line using M-estimator
    cv::Vec3d opt_line(1, 0, 0);
    ceres::Problem problem;
    for (size_t i = 0; i < data.size(); i++)
    {
        ceres::CostFunction* cost_func = new ceres::AutoDiffCostFunction<GeometricError, 1, 3>(new GeometricError(data[i]));
        ceres::LossFunction* loss_func = NULL;
        if (loss_width > 0) loss_func = new ceres::CauchyLoss(loss_width);
        problem.AddResidualBlock(cost_func, loss_func, opt_line.val);
    }
    ceres::Solver::Options options;
    options.linear_solver_type = ceres::ITERATIVE_SCHUR;
    options.num_threads = 8;
    options.minimizer_progress_to_stdout = true;
    ceres::Solver::Summary summary;
    ceres::Solve(options, &problem, &summary);
    std::cout << summary.FullReport() << std::endl;
    opt_line /= sqrt(opt_line[0] * opt_line[0] + opt_line[1] * opt_line[1]); // Normalize
    ...
    return 0;
}
```

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

Overview of Robust Parameter Estimation

- Bottom-up Approaches (~ Voting) e.g. line fitting, relative pose estimation
  - **Hough transform**
    - A datum votes multiple parameter candidates.
      - Note) The parameter space is maintained as a multi-dimensional histogram (discretization).
    - Score: The number of hits by data
    - Selection: Finding a peak on the histogram after voting
  - **RANSAC family**
    - A sample of data votes a single parameter candidate.
    - Score: The number of inlier candidates (whose error is within threshold)
    - Selection: Keeping the best model during RANSAC's iterations
      - Note) RANSAC involves many iterations of parameter estimation and error calculation.

- Top-down Approaches e.g. graph SLAM, multi-view reconstruction
  - **M-estimator**
    - All data aims to find the best parameter (from its initial guess).
    - Score: A cost function
      - The cost function includes a truncated loss function.
    - Selection: Minimizing the cost function (following its gradient)
      - Note) Nonlinear optimization is computationally heavy and leads to a local minima.