RANSAC

ML Active Learning

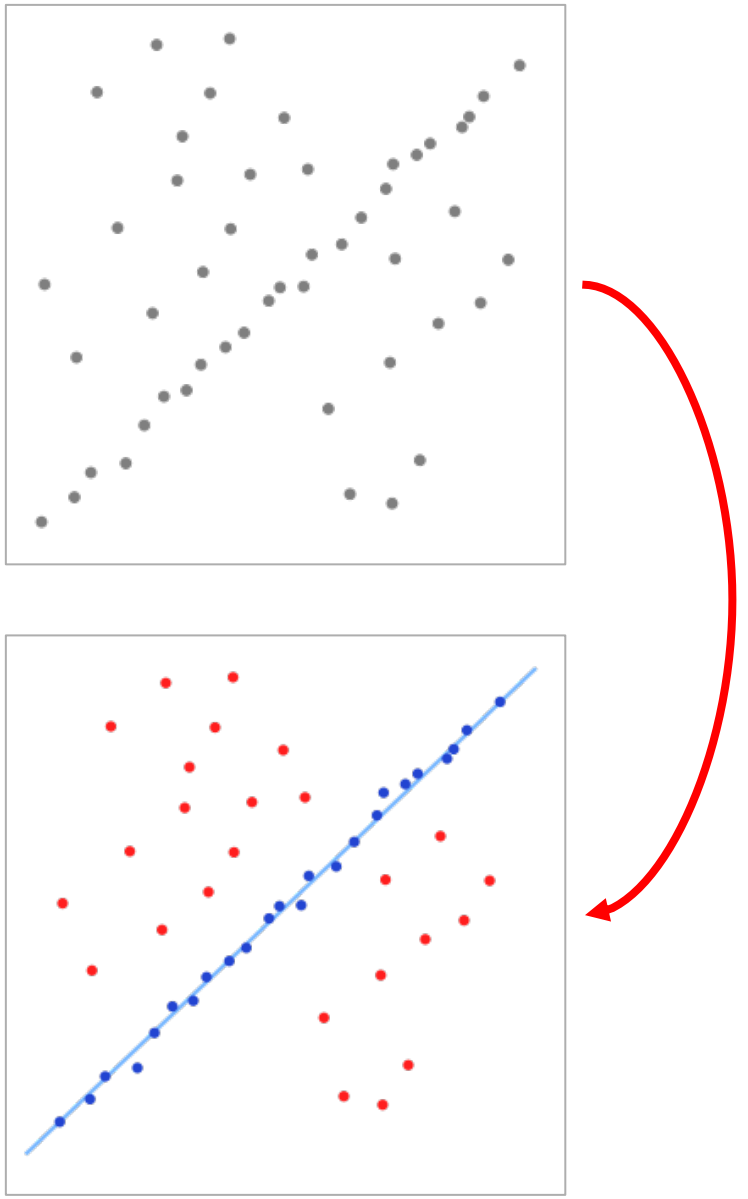
발표자: 202135786 송우석

목차

- What is RANSAC?
- Why we use?
- RANSAC
- Lo-RANSAC
- Lo-RANSAC vs RANSAC
- Application
  - Local Feature matching
- Conclusion
- Reference

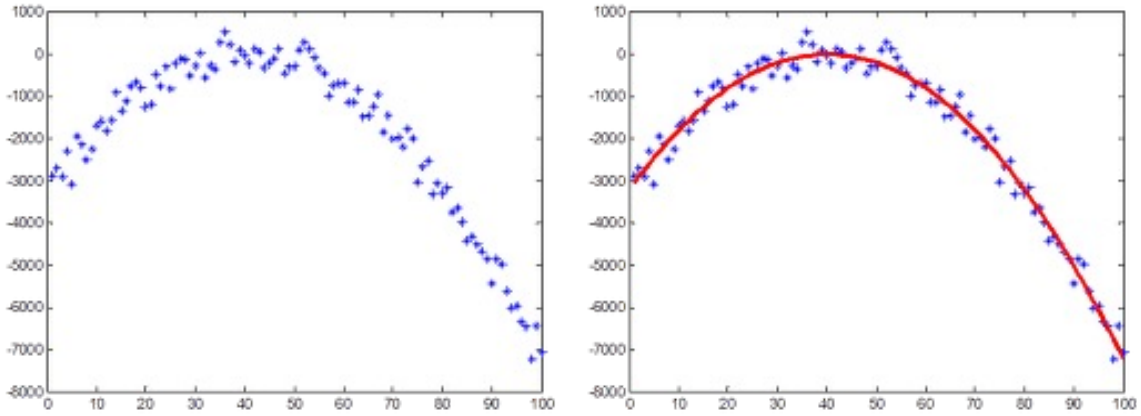
What is RANSAC?

- RANdom SAmple Consensus의 약자
- 데이터에서 outlier를 제거하기 위한 방법론 중 하나
- 크게 데이터 샘플링, 모델 fitting, 목표치 도달, 이렇게 3단계로 구성
- 컴퓨터 비전 분야에서 correspondence problem을 다루거나, fundamental matrix를 추정하기 위해 사용

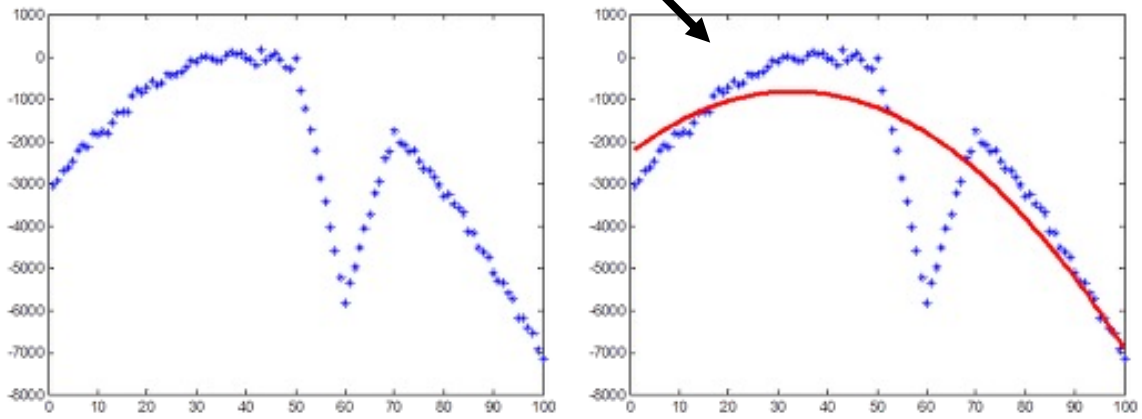


Why we use?

Not fitted well…



Least Square method fitting data without outlier

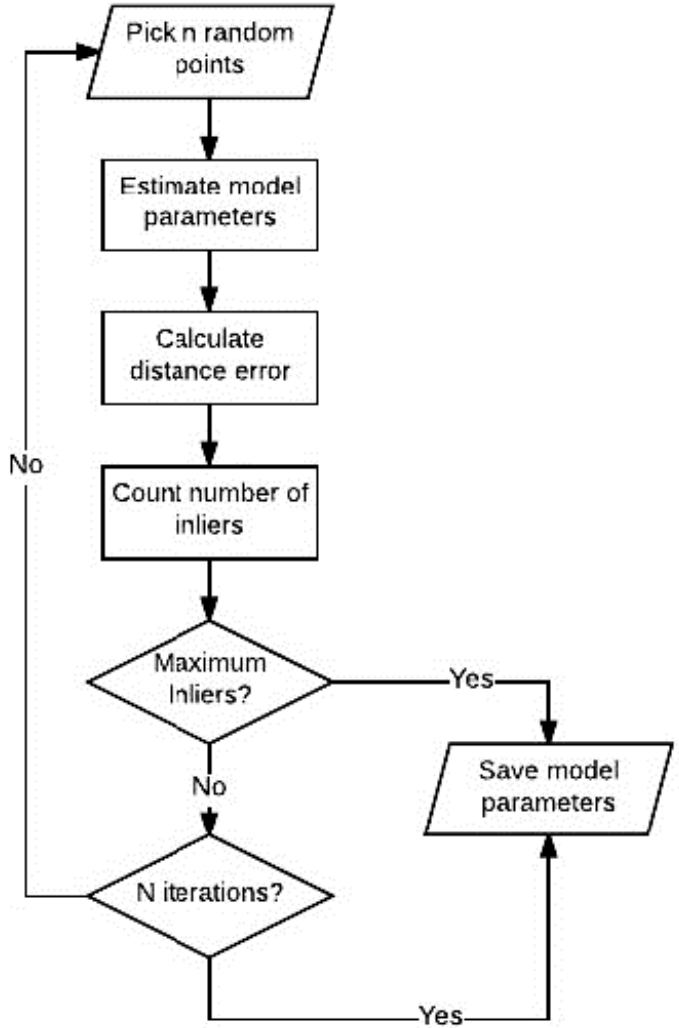


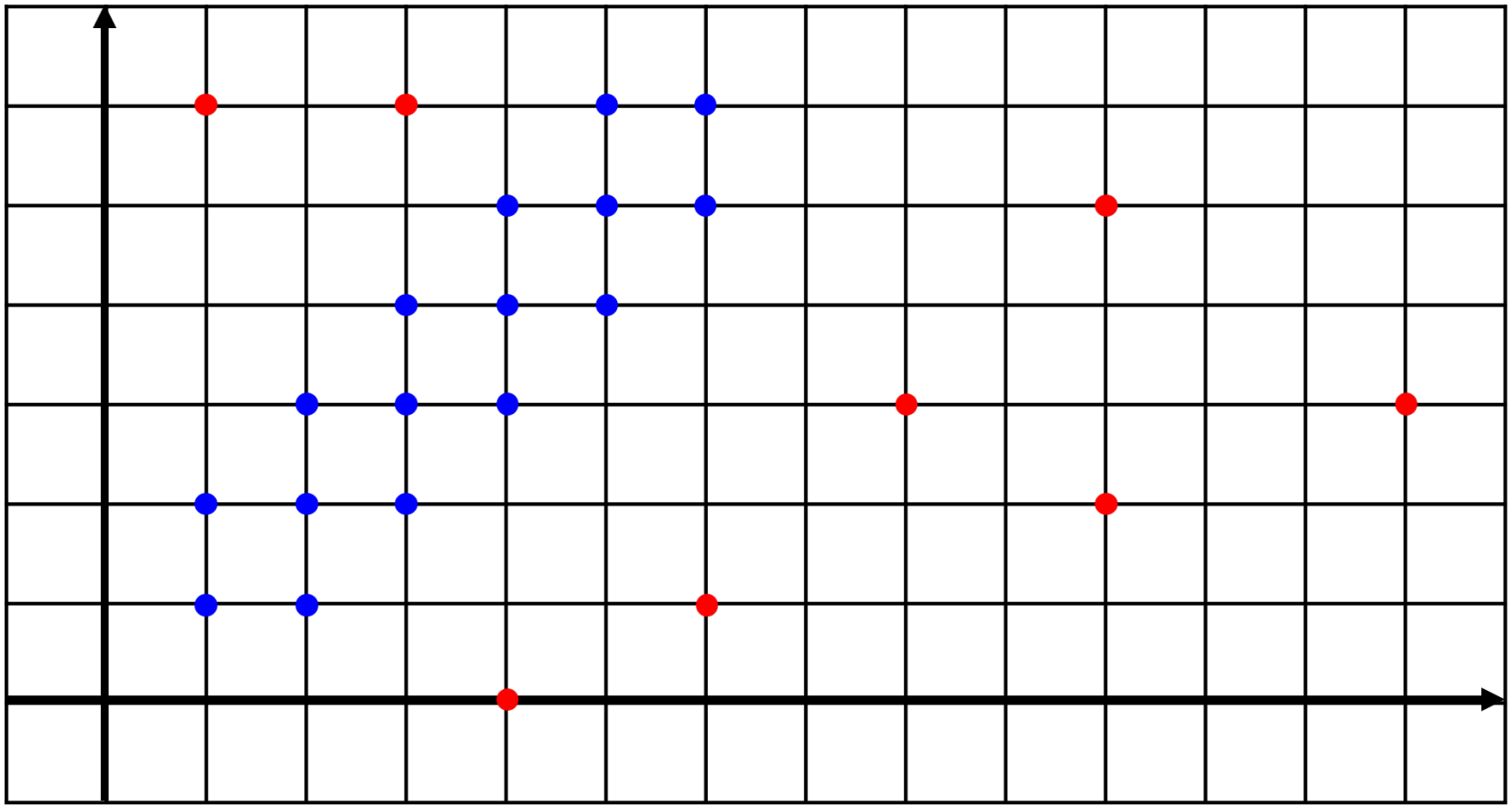
Least Square method fitting data with outlier

- 기존 Least Square는 데이터에 너무 의존 → **oulier에 취약**

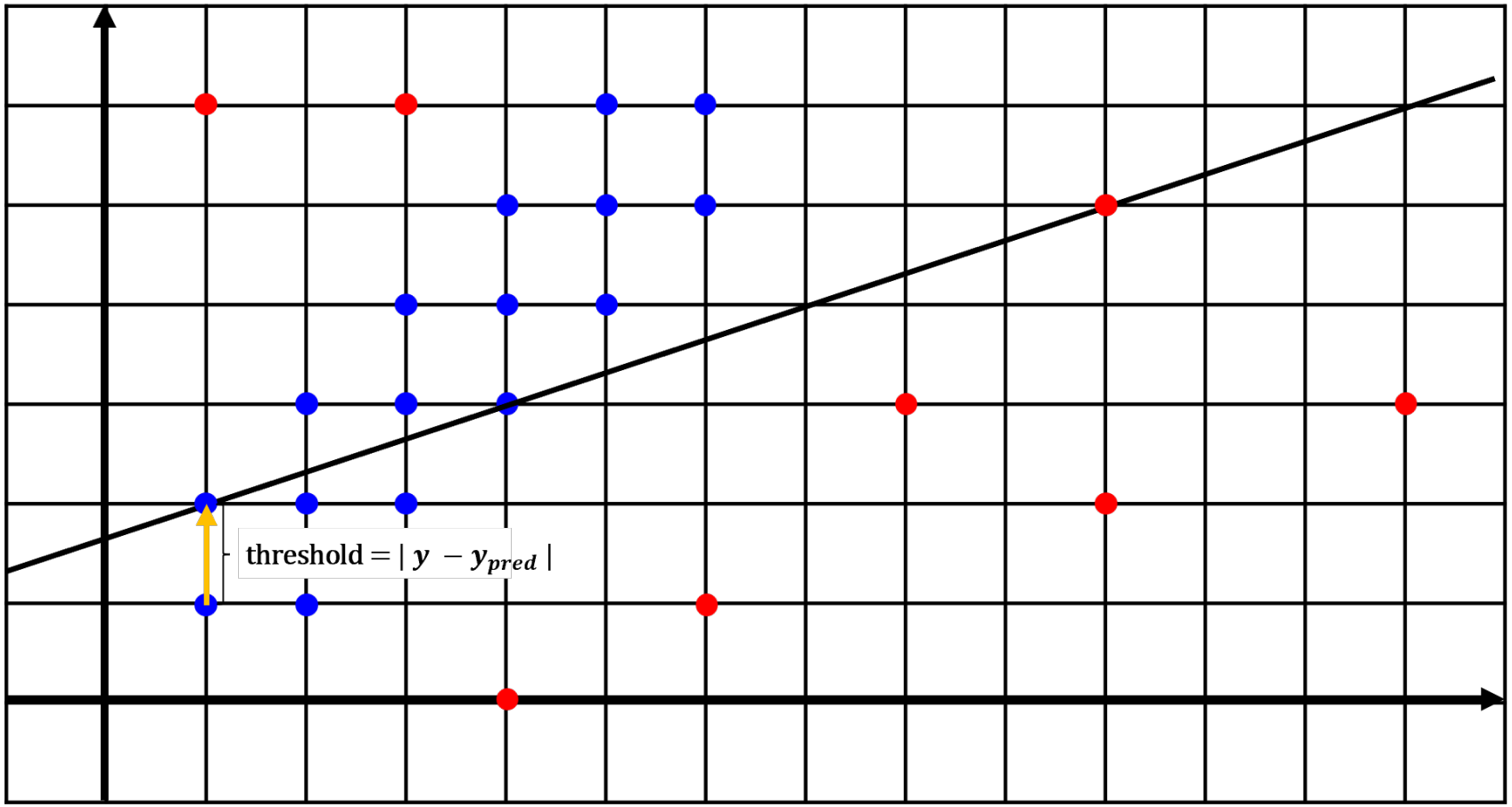
RANSAC

1. 주어진 데이터셋에서 랜덤으로 n개의 점을 샘플링 → sample size
2. 모델 파라미터 추정 → polynomial degree
3. 거리 오차 계산 → threshold
4. inlier 개수 세기
5. 현재 inlier의 개수가 최대 inlier인지 비교
   - 최대 inlier면 파라미터 업데이트
   - 최대 inlier가 아니면 계속 진행
6. N번째 반복이면 끝내기, 아니면 1~5 반복 → sampling number



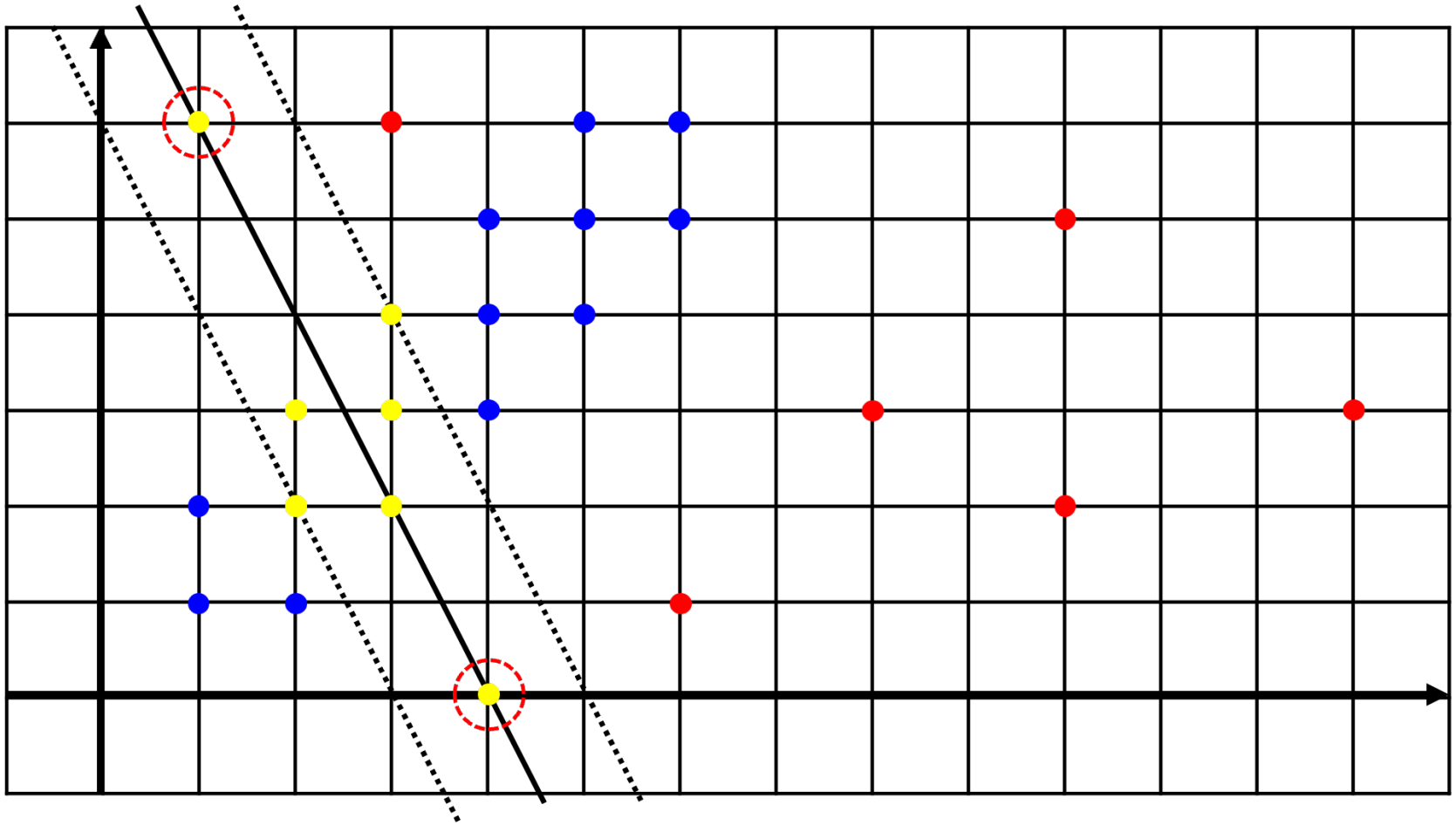


- inlier data
- outlier data



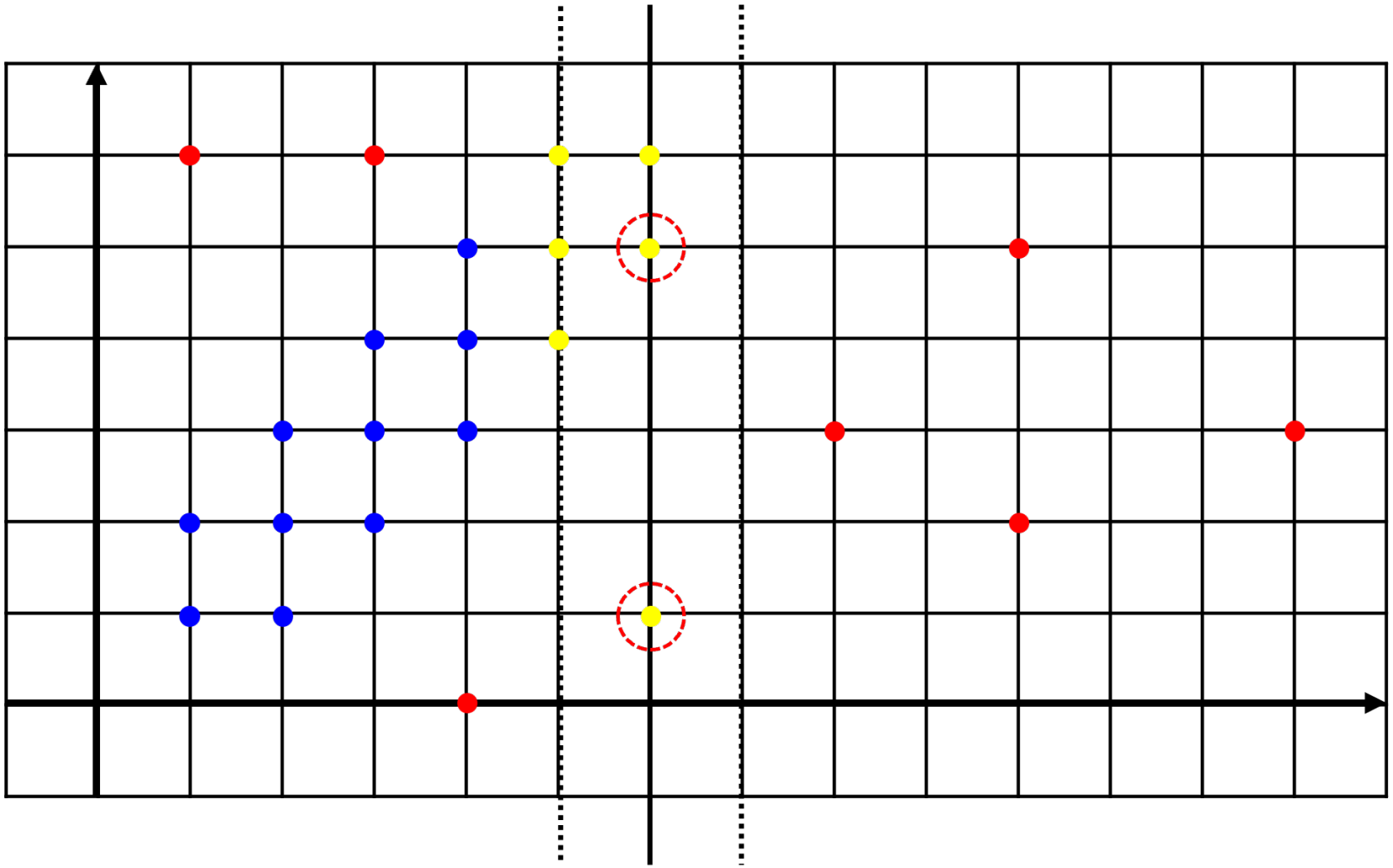
Assume threshold = 1

#inlier : 7, ratio : 29.2 %

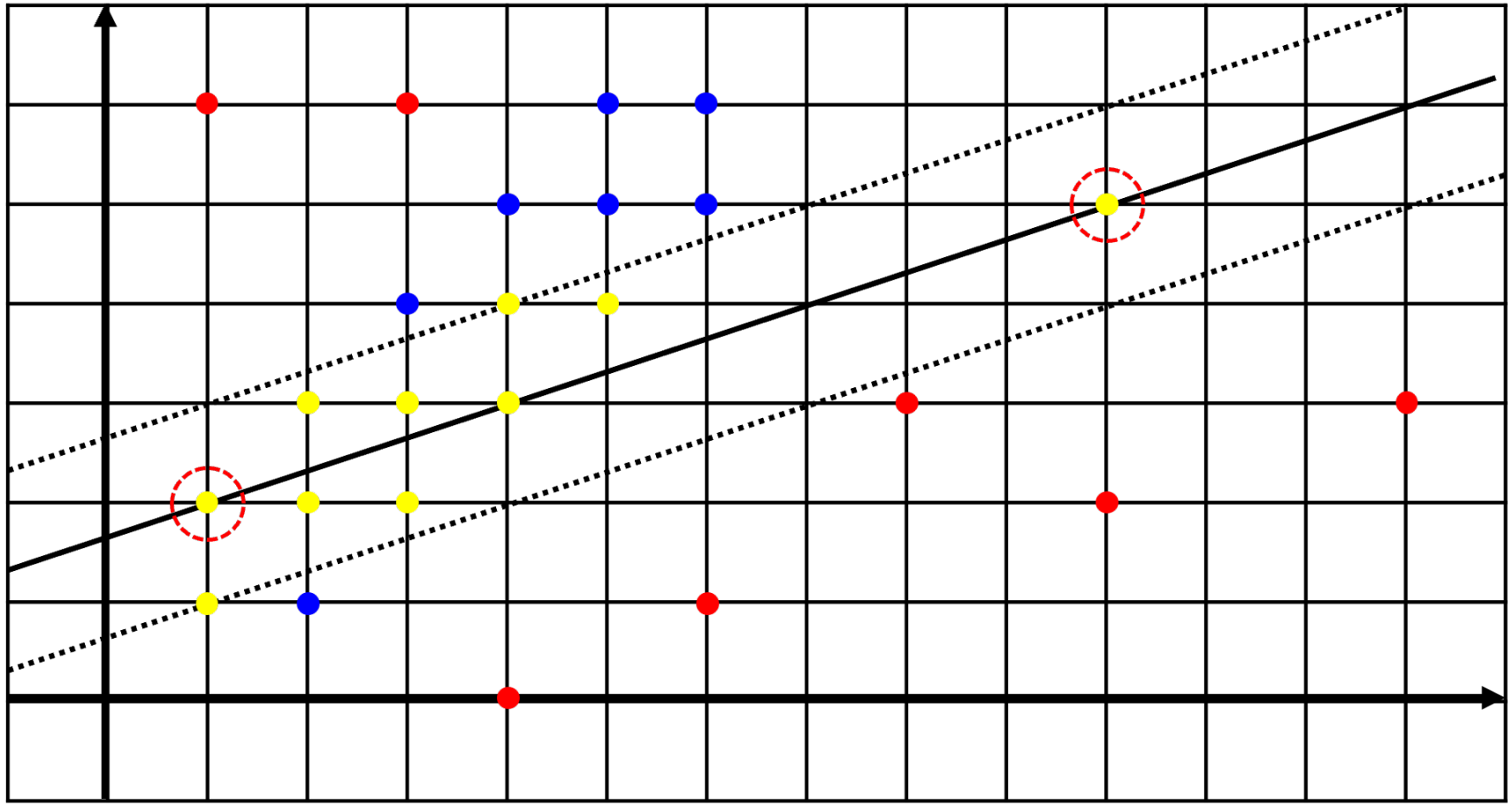


Assume threshold = 1

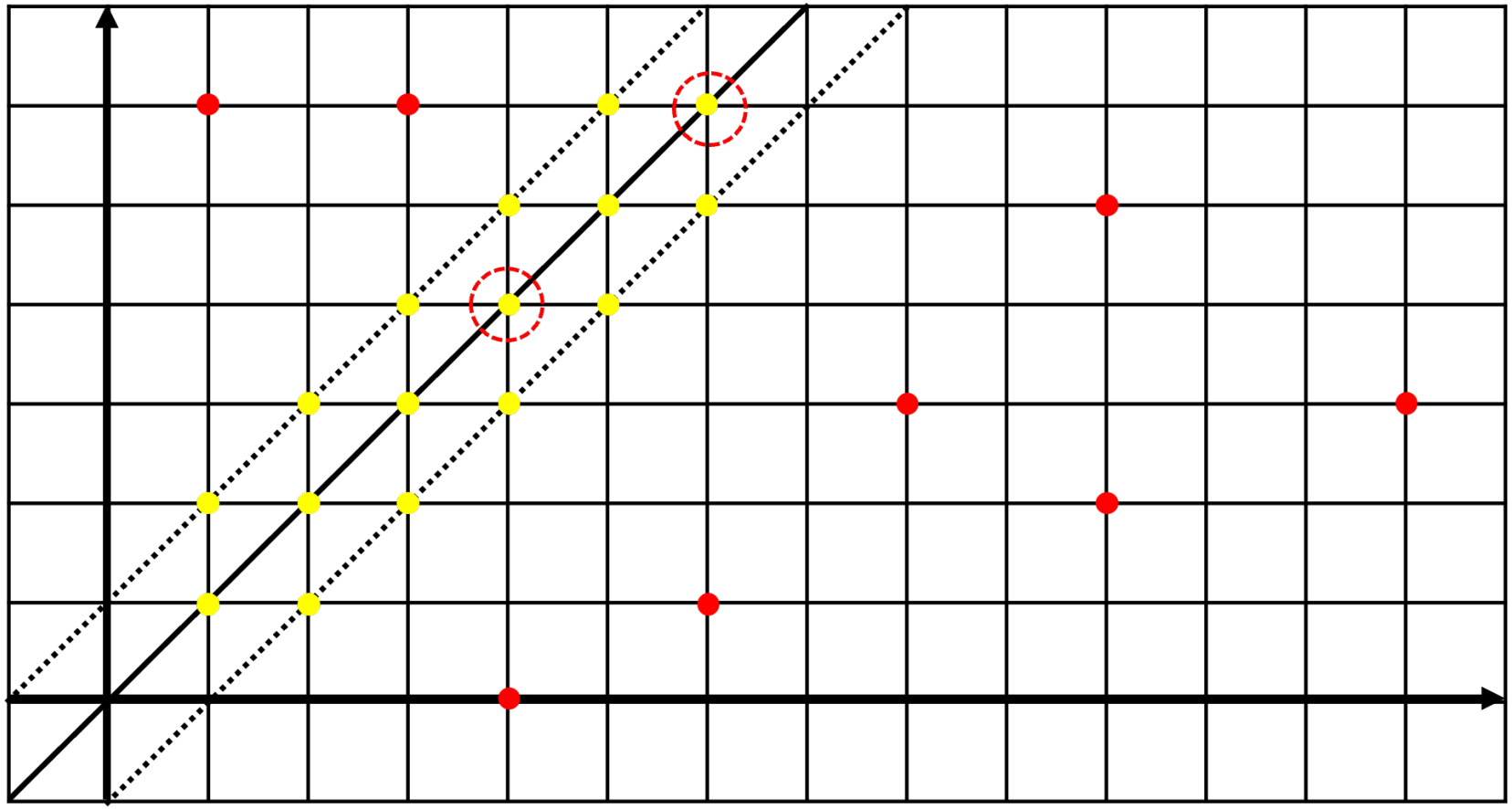
#inlier : 6, ratio : 25 %



#inlier : 10, ratio : 41.7 %



#inlier : 16, ratio : 66.7 %



RANSAC

$$p = 1 - (1 - (1 - e)^m)^N$$

p : Probability of obtaining a sample consisting only of inliers – sampling success

e : ratio of outliers in dataset

m : Number of data sampled per time

N : Number of algorithm iterations

parameter we set (fix)

parameter we'll calculate

$$N = \frac{\log(1-p)}{\log(1-(1-e)^m)} = \frac{\log(1-0.99)}{\log(1-(1-0.5)^m)}$$

binomial distribution

$$P(\text{success, once}) = (1-e)^m$$

$$1 - p = 1 - (1-e)^m$$

sampling N times

$$1 - p = (1 - (1-e)^m)^N$$

$$\therefore \quad p = 1 - (1 - (1-e)^m)^N$$

P(success, N)

RANSAC

- Early Stop
  - RANSAC을 통해 반복적으로 모델을 최적화 할 때, 적합한 모델을 찾으면 더 이상 모델을 fitting 할 필요가 없음

    → 적합한 모델을 찾으면 빨리 끝내자 (early stopping)

    → 보통 앞서 구한 N값을 max iteration으로 설정하여 RANSAC의 반복 횟수 상한을 설정하고, early stop을 위한 inlier ratio를 설정하여 모델 fitting에서 inlier의 비율이 inlier ratio를 초과하면 early stop을 하도록 함

RANSAC

$$N = \frac{\log(1-p)}{\log(1-(1-e)^m)} = \frac{\log(1-0.99)}{\log(1-(1-0.5)^m)}$$

parameter that we handled

calculate N

```python
def get_sampling_number(sample_size, p=0.99, e=0.5):
    '''
    sample size : Number of sample size per every iterations
    p : Desired probability of choosing at least one sample free of outliers
    e : Estimated probability that a point is an outlier
    '''
    # Calculate the required number of iterations based on the formula
    n_iterations_calculated = math.ceil(math.log(1 - p) / math.log(1 - (1 - e)**sample_size))
    print(f"Calculated number of iterations: {n_iterations_calculated}")
    return n_iterations_calculated
```

Goal: find N (sampling number)

for early stopping

to save best threshold

model fitting

find best threshold

```
threshold_cadidates = [1,2,4,8,16,32,64,128]
```

draw best result

return value

Goal: find best threshold

```python
def get_inlier_threshold(thresholds, data, polynomial_degree, sample_size,
                         min_iteration, max_iteration, stop_inlier_ratio, verbose=False):

    early_stop_flag = False
    inlier_threshold = None
    for threshold in thresholds:
        best_fit = None
        best_error = 0
        for i in range(max_iteration):
            # Randomly select sample points
            subset = data[np.random.choice(len(data), sample_size, replace=False)]
            x_sample, y_sample = subset[:, 0], subset[:, 1]

            # Fit a line to the sample points
            p = np.polyfit(x_sample, y_sample, polynomial_degree)

            # Compute error
            y_pred = np.polyval(p, X)
            error = np.abs(y - y_pred)

            # Count inliers
            inliers = error < threshold
            n_inliers = np.sum(inliers)

            # Update best fit if the current model is better
            if n_inliers > best_error:
                print("threshold : {}, index : {}, n_inliers : {}".format(threshold, i, n_inliers))
                best_fit = p
                best_error = n_inliers

                if (i > min_iteration) and (n_inliers/len(data)) >= stop_inlier_ratio:
                    early_stop_flag = True
                    inlier_threshold = threshold

            if early_stop_flag:
                break

        if verbose:
            # Best curve
            y_best = np.polyval(best_fit, X)

            # Plotting
            plt.scatter(X, y, label='Data Points')
            plt.plot(X, y_best, color='red', label='RANSAC Fit')
            plt.legend()
            plt.show()

        if early_stop_flag:
            break

    return inlier_threshold
```

Goal: find best model

for early stopping

to save best model
(coefficient)

to save best inlier

find best model

model
fitting

draw best result

return value

```python
def get_model_with_ransac(data, polynomial_degree, threshold, sample_size,
                          min_iteration, max_iteration, stop_inlier_ratio, verbose=False):
    X = data[:, 0]
    y = data[:, 1]
    early_stop_flag = False
    best_fit = None
    best_error = 0
    for i in range(max_iteration):
        # Randomly select sample points
        subset = data[np.random.choice(len(data), sample_size, replace=False)]
        x_sample, y_sample = subset[:, 0], subset[:, 1]

        # Fit a line to the sample points
        p = np.polyfit(x_sample, y_sample, polynomial_degree)

        # Compute error
        y_pred = np.polyval(p, X)
        error = np.abs(y - y_pred)

        # Count inliers
        inliers = error < threshold
        n_inliers = np.sum(inliers)

        # Update best fit if the current model is better
        if n_inliers > best_error:
            best_fit = p
            best_error = n_inliers
            if (i > min_iteration) and (n_inliers / len(X)) >= stop_inlier_ratio:
                early_stop_flag = True

                print(f"index : {i}, n_inliers : {n_inliers}")
                break

        if early_stop_flag:
            break

    if verbose:
        # Best curve
        y_best = np.polyval(best_fit, X)

        # Plotting
        plt.scatter(X, y, label='Data Points')
        plt.plot(X, y_best, color='red', label='RANSAC Fit')
        plt.legend()
        plt.show()

    return early_stop_flag, best_fit
```

1차 함수 모델링

```python
# Generate synthetic data
np.random.seed(0)
n_points = 100
X = np.linspace(0, 10, n_points)
y = 3 * X + 10 + np.random.normal(0, 3, n_points)

# Add outliers
n_outliers = 20
X[-n_outliers:] += int(30 * np.random.rand())
y[-n_outliers:] -= int(50 * np.random.rand())
X = np.expand_dims(X, -1)
y = np.expand_dims(y, -1)
data = np.hstack([X, y])

threshold_cadidates = [1,2,4,8,16,32,64,128]
threshold_cadidates.sort()

sample_size = 2
max_iteration = get_sampling_number(sample_size)
threshold = get_inlier_threshold(
    threshold_cadidates, data, polynomial_degree=1, sample_size=sample_size,
    min_iteration=-1, max_iteration=max_iteration, stop_inlier_ratio=0.50, verbose=True)

# Print Final parameters
#data, polynomial_degree=1, threshold=threshold, sample_size=sample_size,
  #  min_iteration=-1, max_iteration=max_iteration, stop_inlier_ratio=0.75, verbose=True
print("\nFinal Parameters\n")
print("polynomial_degree: ", 1)
print("threshold: ", threshold)
print("max_iteration: ", max_iteration)
print("stop_inlier_ratio: ", 0.5)
```

total number of data

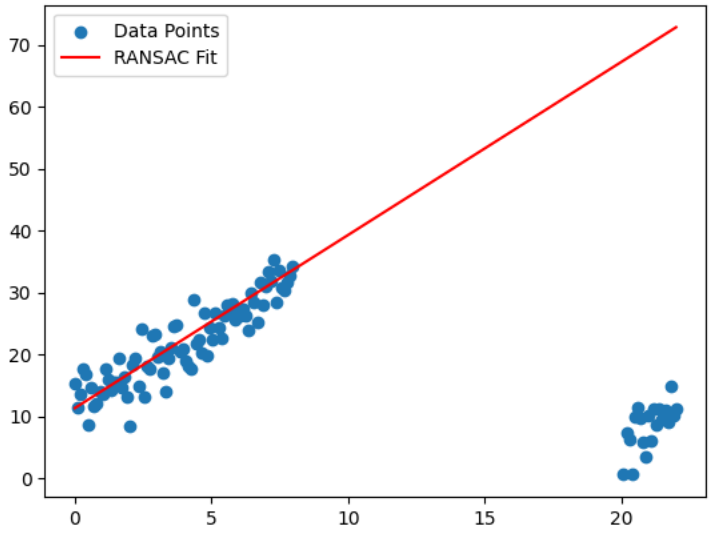
number of outliers

candidates of threshold to find best threshold

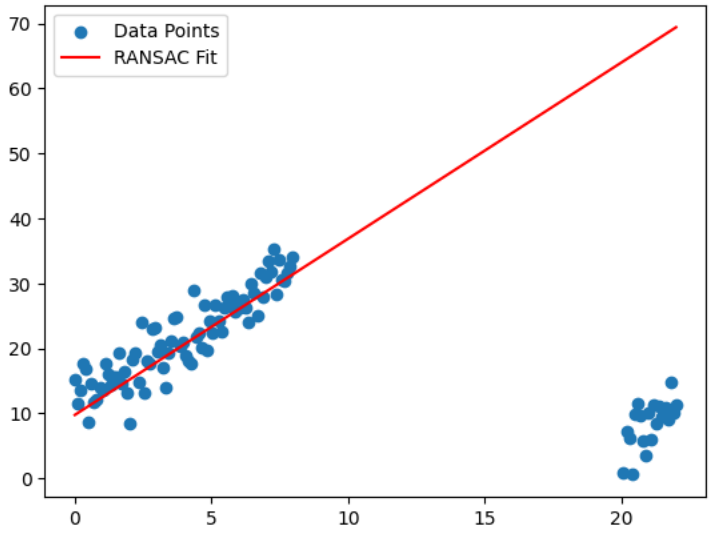
sample size

get sampling number

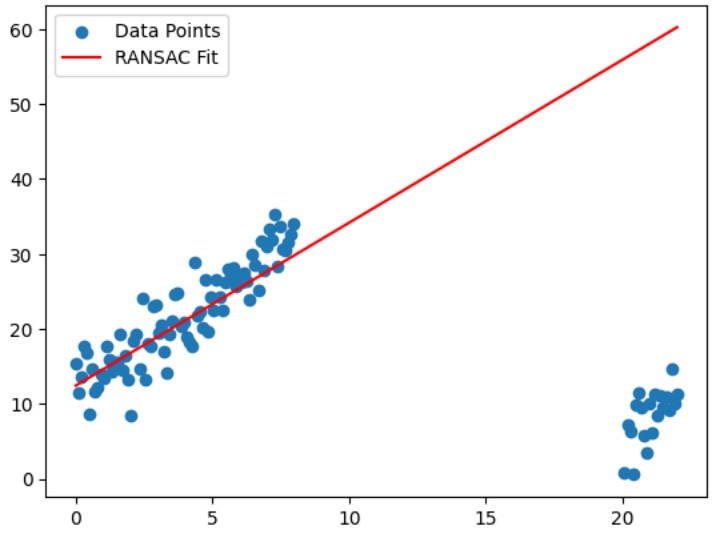
get threshold




```
Calculated number of iterations: 17
threshold : 1, index : 0, n_inliers : 16
threshold : 1, index : 10, n_inliers : 21
```




```
threshold : 2, index : 0, n_inliers : 16
threshold : 2, index : 1, n_inliers : 18
threshold : 2, index : 2, n_inliers : 29
threshold : 2, index : 4, n_inliers : 35
threshold : 2, index : 9, n_inliers : 40
```




```
threshold : 4, index : 0, n_inliers : 2
threshold : 4, index : 1, n_inliers : 41
threshold : 4, index : 3, n_inliers : 43
threshold : 4, index : 4, n_inliers : 63
```

How about polynomial data?

best threshold

the most inliers

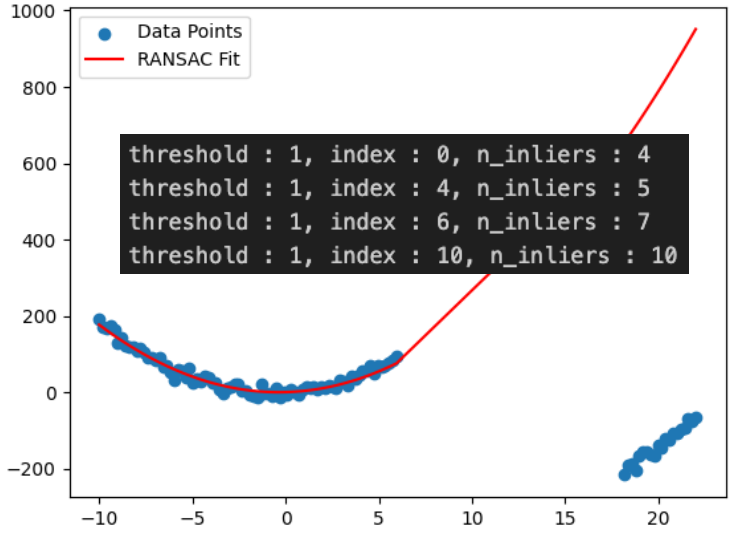
2차 함수 모델링

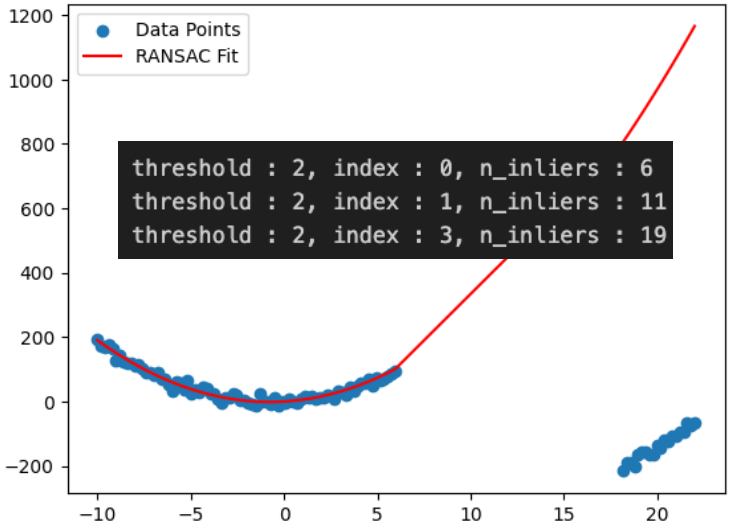
```python
import numpy as np
import math
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(0)
n_points = 100
X = np.linspace(-10, 10, n_points)
y = 2 * X**2 + 3 * X + 4 + np.random.normal(0, 10, n_points)

# Add outliers
n_outliers = 20
X[-n_outliers:] += int(30 * np.random.rand())
y[-n_outliers:] -= int(500 * np.random.rand())

X = np.expand_dims(X, -1)
y = np.expand_dims(y, -1)
data = np.hstack([X, y])

threshold_cadidates = [1,2,4,8,16,32,64,128]
threshold_cadidates.sort()

sample_size = 3
max_iteration = get_sampling_number(sample_size)
threshold = get_inlier_threshold(
    threshold_cadidates, data, polynomial_degree=2, sample_size=sample_size,
    min_iteration=-1, max_iteration=max_iteration, stop_inlier_ratio=0.50, verbose=True)
```

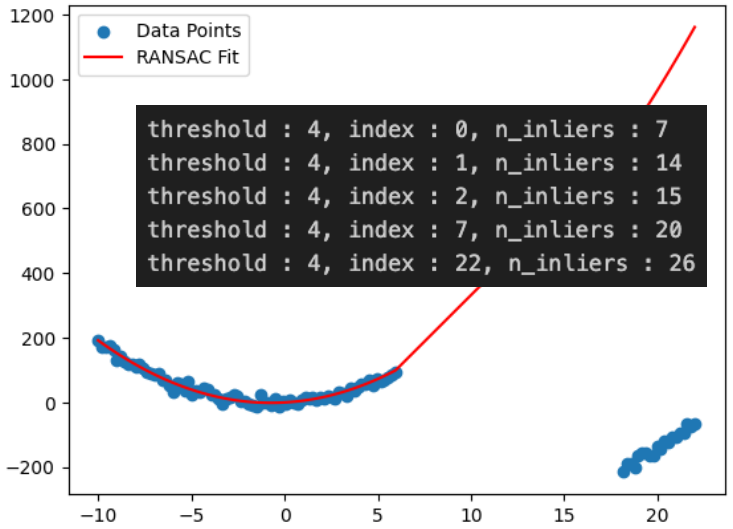
change sample size to 3

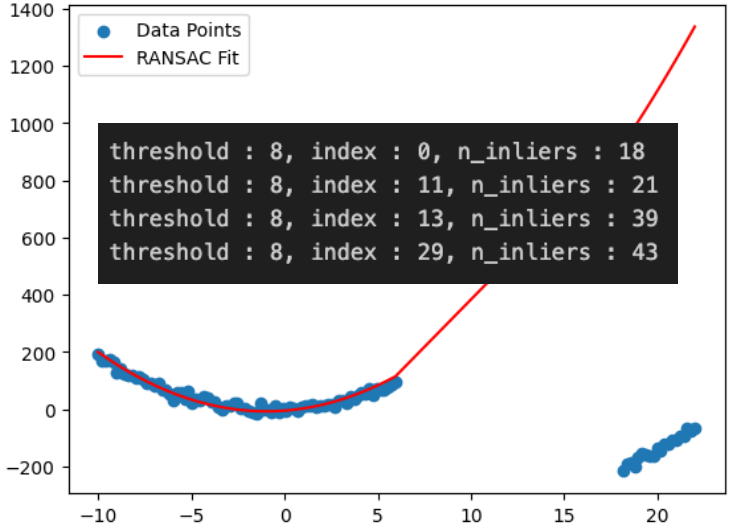
change degree to 2

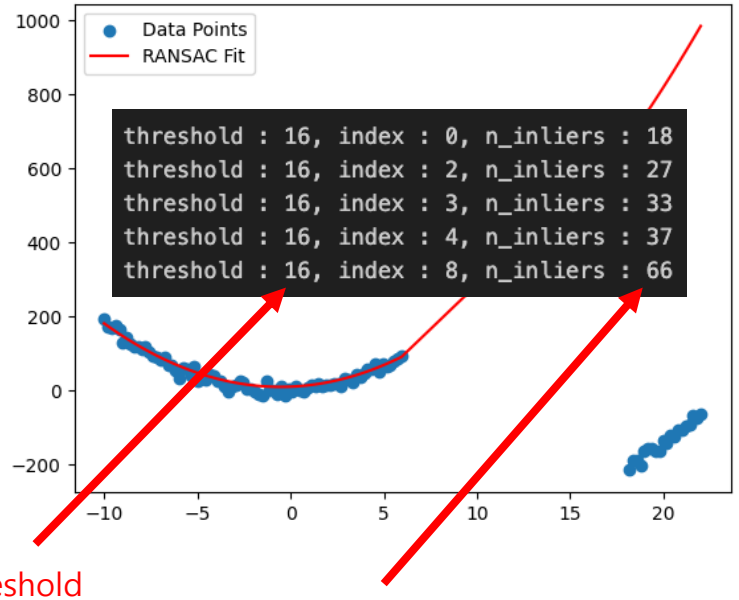
Similar as before


Data Points
RANSAC Fit
threshold : 1, index : 0, n_inliers : 4
threshold : 1, index : 4, n_inliers : 5
threshold : 1, index : 6, n_inliers : 7
threshold : 1, index : 10, n_inliers : 10


Data Points
RANSAC Fit
threshold : 2, index : 0, n_inliers : 6
threshold : 2, index : 1, n_inliers : 11
threshold : 2, index : 3, n_inliers : 19


Data Points
RANSAC Fit
threshold : 4, index : 0, n_inliers : 7
threshold : 4, index : 1, n_inliers : 14
threshold : 4, index : 2, n_inliers : 15
threshold : 4, index : 7, n_inliers : 20
threshold : 4, index : 22, n_inliers : 26


Data Points
RANSAC Fit
threshold : 8, index : 0, n_inliers : 18
threshold : 8, index : 11, n_inliers : 21
threshold : 8, index : 13, n_inliers : 39
threshold : 8, index : 29, n_inliers : 43


Data Points
RANSAC Fit
threshold : 16, index : 0, n_inliers : 18
threshold : 16, index : 2, n_inliers : 27
threshold : 16, index : 3, n_inliers : 33
threshold : 16, index : 4, n_inliers : 37
threshold : 16, index : 8, n_inliers : 66
best threshold
the most inliers

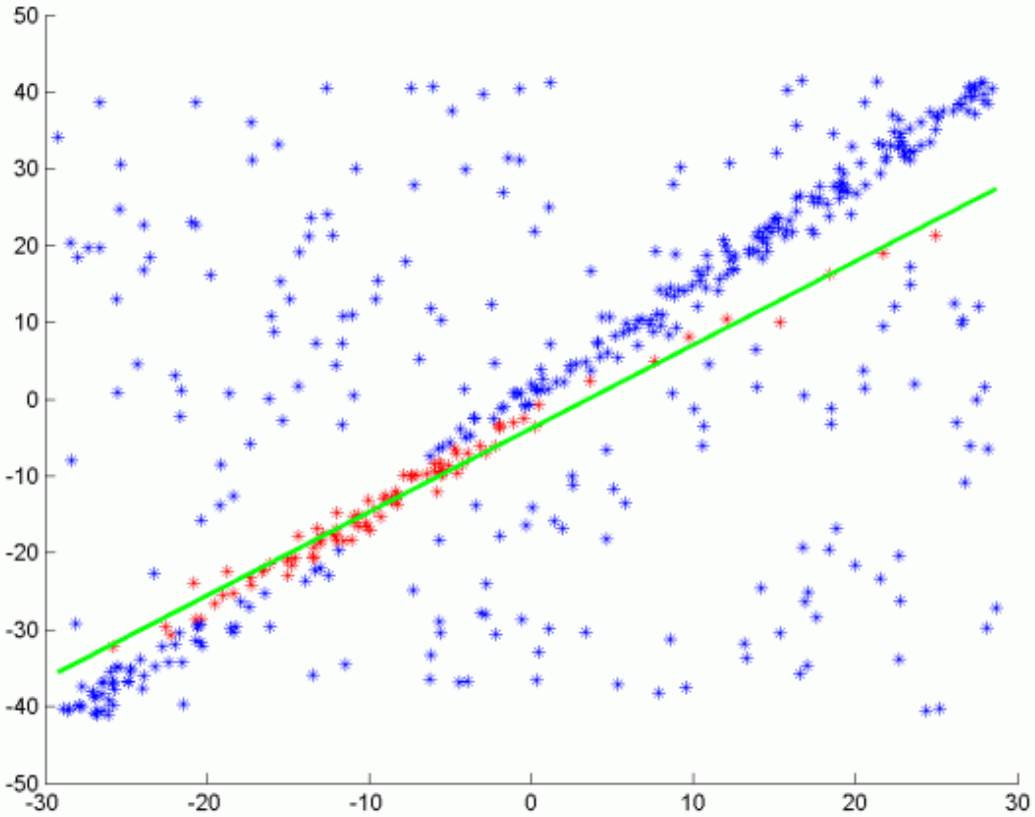
RANSAC

장점

- outlier에 강건한 모델을 만들 수 있음
- 일찍 끝나면 그 만큼 시간을 아낌
  - better for real-time program
- 이해하기 쉬움

단점

- Non-deterministic
- threshold 파라미터에 민감
  - threshold ↓: 데이터 변화에 민감해짐
  - threshold ↑: outlier를 inlier로 착각
- Loss 기반 최적화 방법이 아님
  - loss function과 달리 pipeline에 한 번에 넣고 학습 불가능



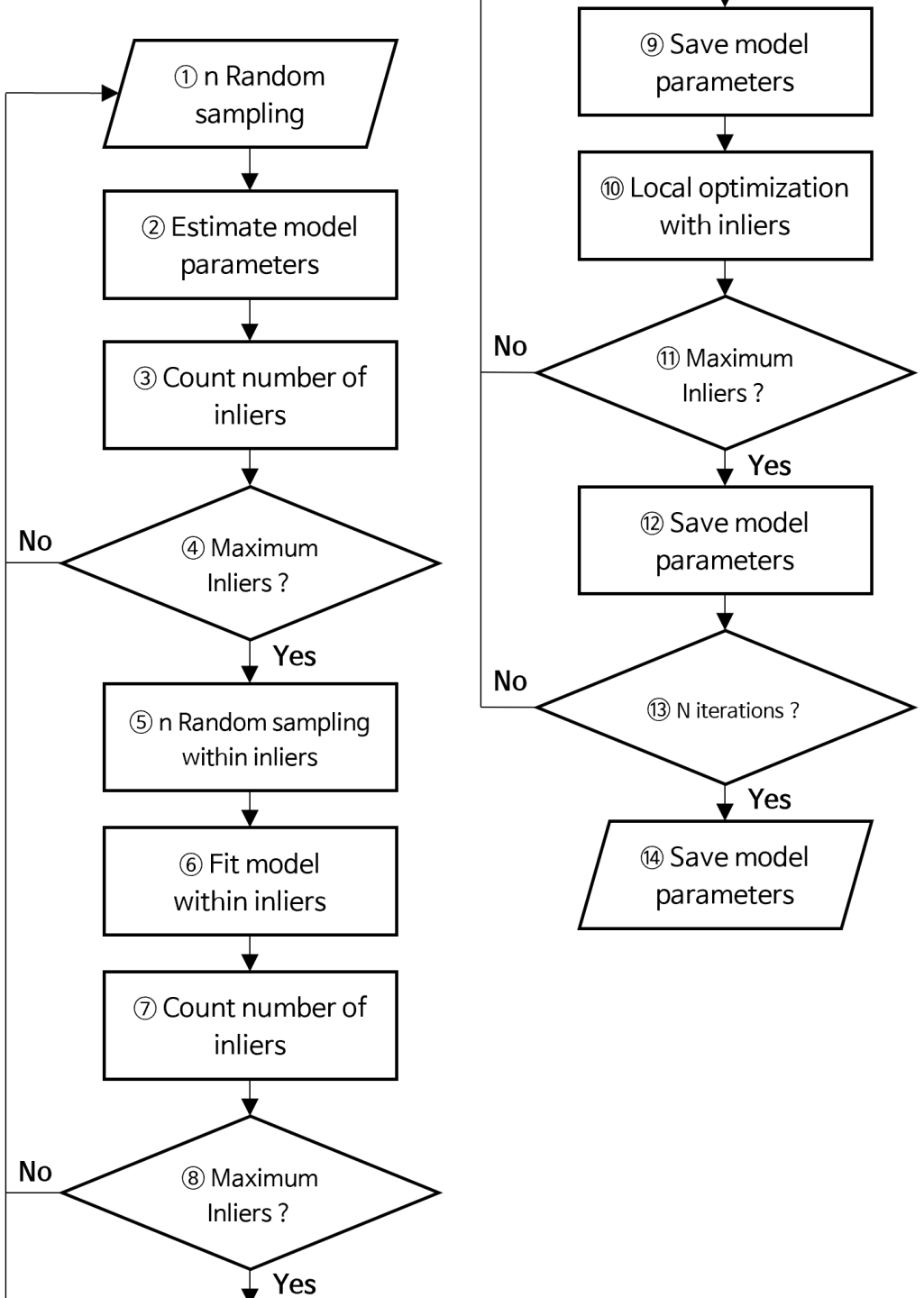
Lo-RANSAC

- Locally Optimized RANSAC
- 'Inlier data는 Inlier data 주변에 모인다'는 특성 이용
  - 최대한 정답에서 가까운 모델을 추정
- 기존 RANSAC과 다르게 현재 모델의 inlier의 개수가 maximum inlier를 달성하면, 현재 inlier의 내부에서 랜덤 샘플링 진행
  - 완전히 랜덤한 샘플을 고르는 기존 RANSAC보다 효율적
  - 신뢰도 높아진 데이터에 대해 local optimization 적용 가능

    e.g. non-linear least square, Levenberg-Marquardt 등

Lo-RANSAC

1. 주어진 데이터셋에서 랜덤으로 n개의 점을 샘플링
2. 모델 파라미터 추정
3. 거리 오차 계산
4. 현재 inlier의 개수가 최대 inlier인지 비교
   - 최대 inlier면 5번으로 이동
   - 최대 inlier가 아니면 1번으로 다시 돌아가기
5. 현재 inlier들에 대해 랜덤으로 n개의 점 샘플링
6. 모델 파라미터 추정
7. inlier의 개수 세기
8. 현재 inlier의 개수가 최대 inlier인지 비교
   - 최대 inlier면 9번으로 이동
   - 최대 inlier가 아니면 1번으로 다시 돌아가기
9. 모델 파라미터 저장
10. 현재 inlier들에서 local optimization 수행
11. 현재 inlier의 개수가 최대 inlier인지 비교
    - 최대 inlier면 12번으로 이동
    - 최대 inlier가 아니면 1번으로 다시 돌아가기
12. N번째 반복인지 확인
    - 맞으면 모델 파라미터 저장 후 종료
    - 아니면 1번으로 다시 돌아가기



```python
# Compute the number of iterations required for RANSAC
def get_sampling_number(sample_size, p=0.99, e=0.5):
    n_iterations_calculated = math.ceil(math.log(1 - p) / math.log(1 - (1 - e)**sample_size))
    print(f"Calculated number of iterations: {n_iterations_calculated}")
    return n_iterations_calculated

# Polynomial evaluation function
def polynomial(x, *coeffs):
    return np.polyval(coeffs, x)

# Fit polynomial using curve fitting
def fit_polynomial(subset, degree):
    x, y = subset[:, 0], subset[:, 1]
    params, _ = curve_fit(lambda x, *coeffs: polynomial(x, *coeffs), x, y, p0=[1]*(degree+1))
    return params

# Count inliers based on residual threshold
def count_inliers(model, data, threshold):
    x, y = data[:, 0], data[:, 1]
    y_pred = np.polyval(model, x)
    return np.where(np.abs(y - y_pred) < threshold)[0]

# Local optimization using Levenberg–Marquardt (LM)
def residuals_poly(coeffs, x, y):
    return (y - np.polyval(coeffs, x)).ravel()

def local_optimization(model, inliers, degree):
    x = inliers[:, 0].ravel()
    y = inliers[:, 1].ravel()
    n_params = degree + 1
    if len(x) < n_params:
        return fit_polynomial(inliers, degree)
    x0 = model if model is not None else fit_polynomial(inliers, degree)
    try:
        res = least_squares(
            residuals_poly,
            x0=x0,
            args=(x, y),
            method='lm',
            max_nfev=200
        )
        return res.x
    except Exception:
        return fit_polynomial(inliers, degree)
```

calculate N (sampling number)

model fitting

count number of inliers

local optimization
(Levenberg-Marquardt)

to save best model
(coefficient)

to save max inlier

total number of data

step 1 ~ 4

step 5 ~ 9

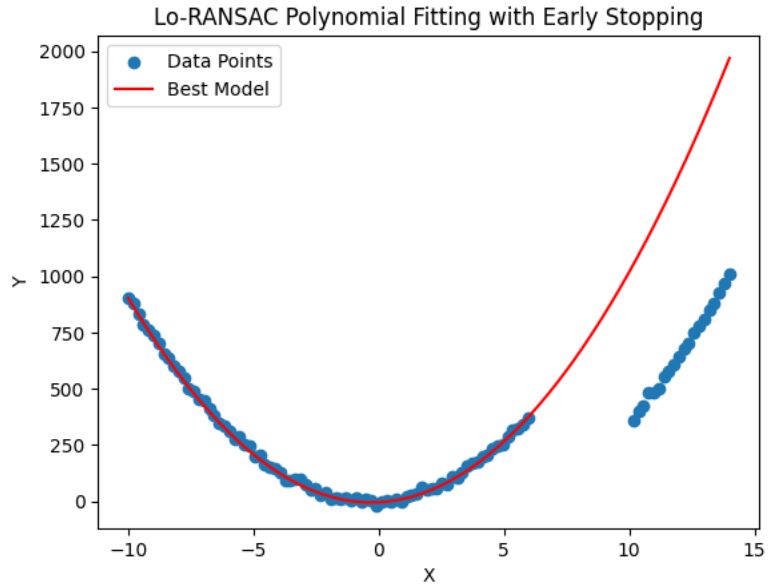
step 10 ~ 11

for early stopping

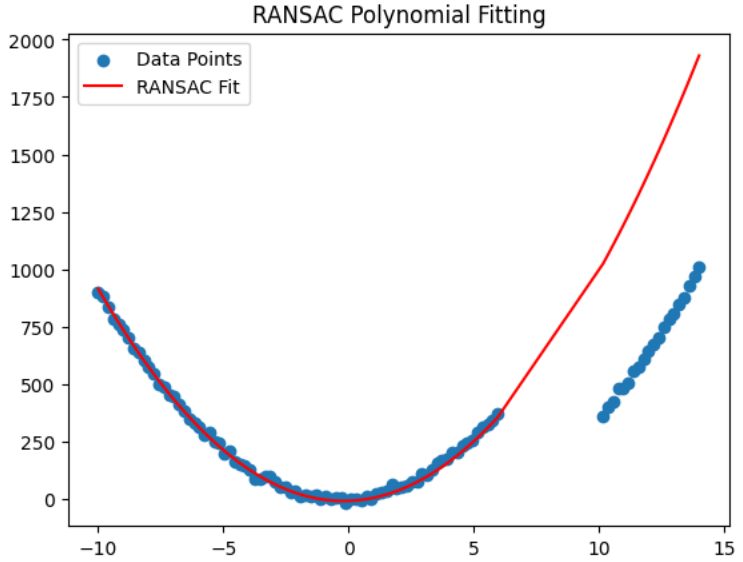
return value

```python
# Lo-RANSAC main loop with LM-based local refinement
def lo_ransac_polynomial(data, degree=2, min_iterations=-1, max_iterations=100, threshold=1.0, stop_inlier_ratio=0.9):
    best_model = None
    max_inliers = 0
    total_data = len(data)

    for i in range(max_iterations):
        # Step 1: Random Sampling
        subset = data[np.random.choice(total_data, degree+1, replace=False)]
        # Step 2: Fit Initial Model
        model = fit_polynomial(subset, degree)
        # Step 3: Count Inliers
        inliers_idx = count_inliers(model, data, threshold)
        inliers = data[inliers_idx]
        # Step 4: Check Initial Model
        if len(inliers) < max_inliers:
            continue

        # Step 5: Random Sampling within inliers
        inner_subset = inliers[np.random.choice(len(inliers), degree+1, replace=False)]
        # Step 6: Fit Model within inliers
        inner_model = fit_polynomial(inner_subset, degree)
        # Step 7: Count Inliers
        inner_inliers_idx = count_inliers(inner_model, data, threshold)
        inner_inliers = data[inner_inliers_idx]
        # Step 8, 9: Check Inner RANSAC Model and Save Best Model
        if len(inner_inliers) > max_inliers:
            best_model = inner_model
            max_inliers = len(inner_inliers)
        else:
            continue

        # Step 10: Local Optimization with inliers (LM refinement)
        refined_model = local_optimization(inner_model, inner_inliers, degree)
        refined_inliers = count_inliers(refined_model, data, threshold)

        # Step 11: Check Local Optimization and Save Best Model
        if len(refined_inliers) > max_inliers:
            best_model = refined_model
            max_inliers = len(refined_inliers)

        # Early stopping condition based on inlier ratio
        inlier_ratio = max_inliers / total_data
        if i >= min_iterations and inlier_ratio >= stop_inlier_ratio:
            print(f"Early stopping at iteration {i}, inlier ratio: {inlier_ratio}")
            break

    print(f"Result : iteration {i}, inlier ratio: {inlier_ratio}")
    return best_model
```

Lo-RANSAC vs RANSAC




```
Calculated number of iterations: 35
Early stopping at iteration 1, inlier ratio: 0.71
Result : iteration 1, inlier ratio: 0.71
```




```
Result : iteration 34, inlier ratio: 0.63
No Early Stop
```

Lo-RANSAC vs RANSAC

Algorithm 1 LO-RANSAC [6]. Example: 8-point RANSAC

1: **for** $k = 1 \rightarrow \mathrm{K}(|\mathcal{I}^*|, \eta)$ **do**
2: $\mathcal{S}_k \leftarrow$ randomly drawn minimal sample — randomly select 8 points
3: $M_k \leftarrow$ model estimated from sample $\mathcal{S}_k$ — compute fundamental matrix F
4: $\mathcal{I}_k \leftarrow find\_inliers(M_k, \theta)$ — find the number of inliers
5: **if** $|\mathcal{I}_k| > |\mathcal{I}_s^*|$ **then**
6: $M_s^* \leftarrow M_k; \mathcal{I}_s^* \leftarrow \mathcal{I}_k$
7: $M_{LO}, \mathcal{I}_{LO} \leftarrow$ run Local Optimization (Alg. 2)
8: **if** $|\mathcal{I}_{LO}| > |\mathcal{I}^*|$ **then**
9: $M^* \leftarrow M_{LO}; \mathcal{I}^* \leftarrow \mathcal{I}_{LO}$
10: update K
11: **end if**
12: **end if**
13: **end for**
14: **return** M^* — the fundamental matrix F which has the most inliers

worst: O(N·log N)

$$\sum_{1}^{n} \frac{1}{n} \leq \int_{1}^{n} \frac{1}{n} dx + 1 = \log\ n + 1$$

= T (반복 횟수)

$nData \leftarrow$ number of data points
$nP \leftarrow$ smallest number of points required by the model
$nI \leftarrow$ number of iterations
$t \leftarrow$ maximum distance (threshold) of a point to the model to be an inlier
$bestInliers \leftarrow 0$;
$bestModel \leftarrow NULL$;
while *Not all iterations done* **do**
  Draw nP points randomly;
  Fit a model M to those points;
  $nInliers \leftarrow 0$;
  for *each point in the data* **do**
    $d \leftarrow$ distance from the point to the model;
    if $d < t$ **then**
      $nInliers \leftarrow nInliers + 1$
  if $nInliers > bestInliers$ **then**
    $bestInliers \leftarrow nInliers$;
    $bestModel \leftarrow M$
return *bestModel*

worst: O(N·T)

$$nI = \frac{log\,(\alpha)}{log\,(1 - iRatio^{nP})}$$

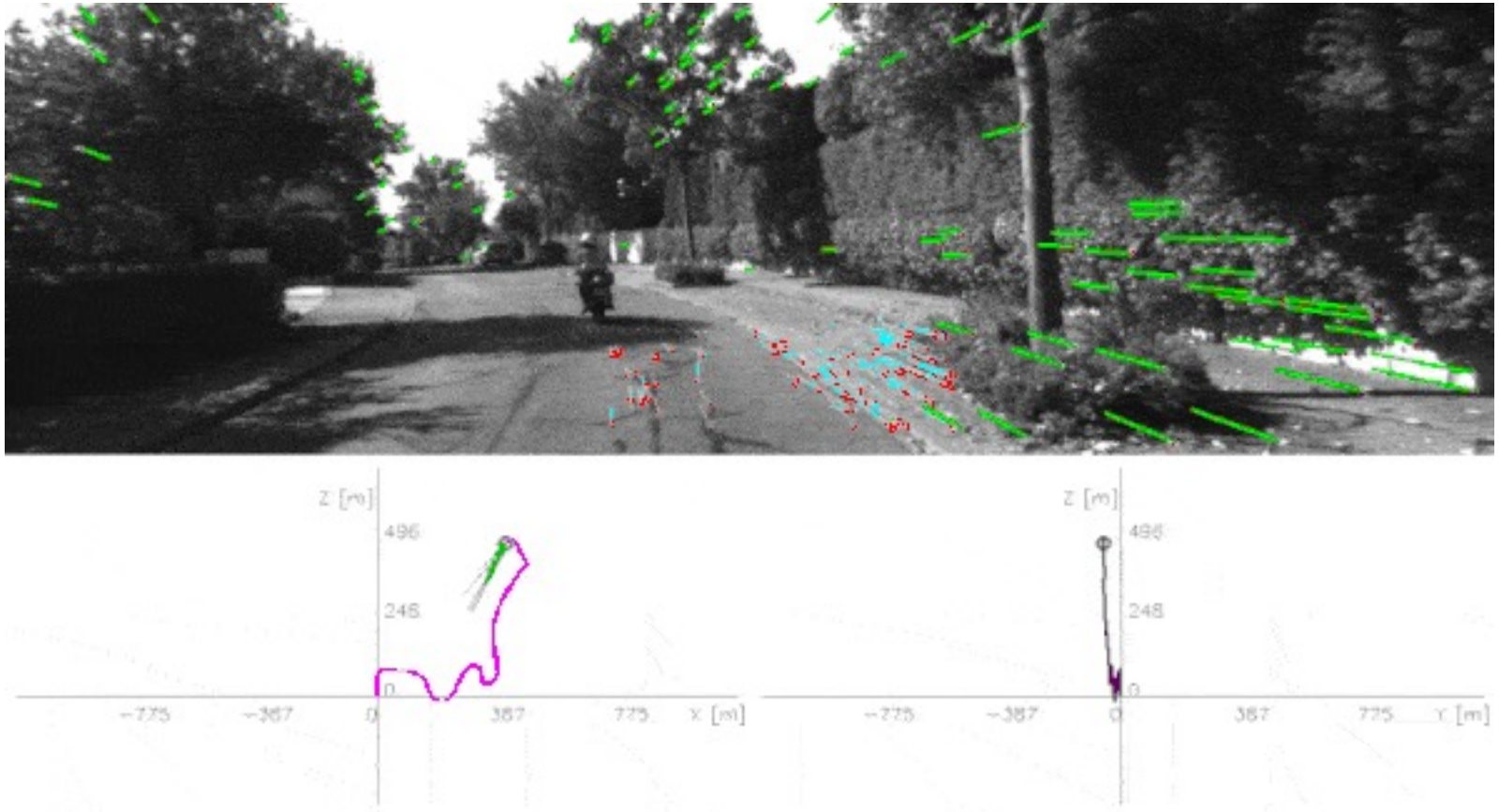
= T (반복 횟수)

Application

- Local feature matching

Application

- visual odometry



Application

- Local feature matching example

Find feature and Match!

```python
# Load images
img1_color = cv2.imread('/Users/song-useog/Desktop/3학년 2학기/머신러닝/active learning/images/cat_1.png')
img2_color = cv2.imread('/Users/song-useog/Desktop/3학년 2학기/머신러닝/active learning/images/cat_2.png')

if img1_color is None or img2_color is None:
    print("Could not open or find the images.")
    exit()

# Make Grayscale images with original images (save original images to visualize)
img1_gray = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
img2_gray = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)

# detect ORB feature and compute descriptor
orb = cv2.ORB_create()
kp1, des1 = orb.detectAndCompute(img1_gray, None)
kp2, des2 = orb.detectAndCompute(img2_gray, None)
```

prepare data

FM_RANSAC Python: cv.FM_RANSAC	RANSAC algorithm. It needs at least 15 points. 7-point algorithm is used.

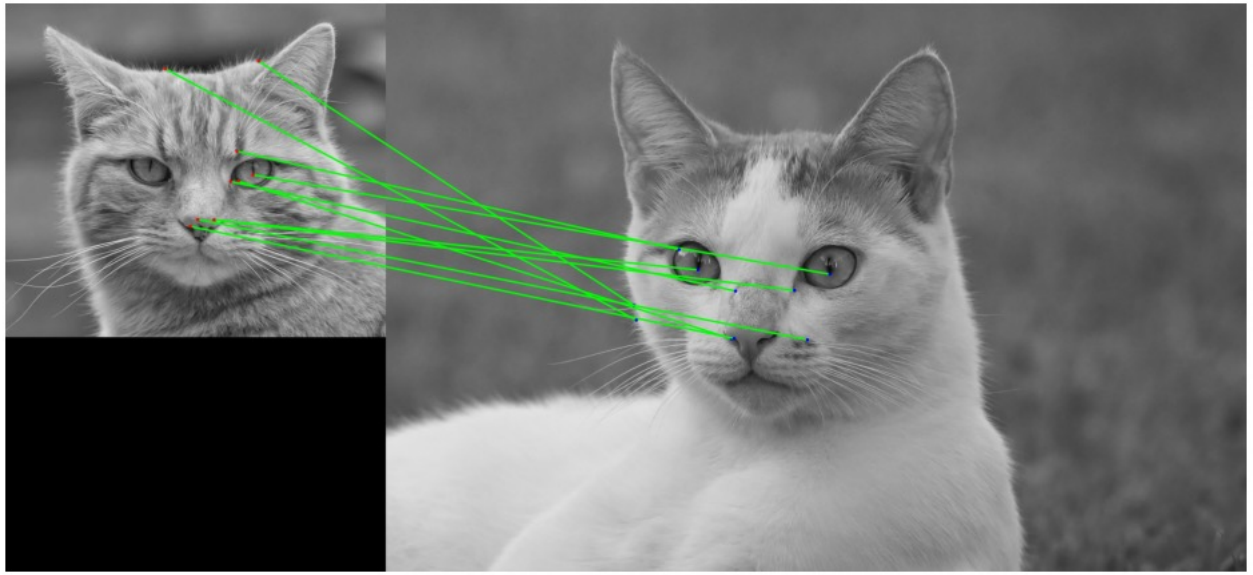
with RANSAC

```python
# Brute-Force matching (Hamming distance)
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key=lambda x: x.distance)
```

without RANSAC

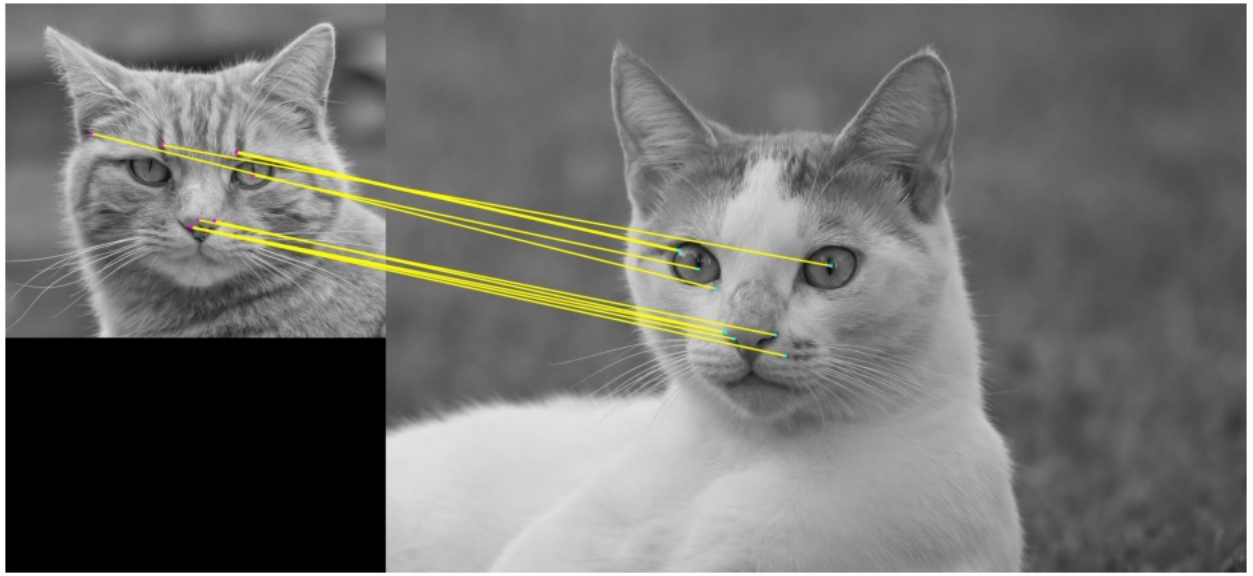
```python
# with RANSAC
if len(matches) >= 8:
    pts1 = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
    pts2 = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

    F, mask = cv2.findFundamentalMat(pts1, pts2, cv2.FM_RANSAC, 20, 0.99)
    if F is not None and mask is not None:
        matchesMask = mask.ravel().tolist()

        # extract TOP 10 inliers
        inlier_matches = [m for i, m in enumerate(matches) if matchesMask[i]]
        inlier_top10 = sorted(inlier_matches, key=lambda x: x.distance)[:10]
```

Brute Force Matching (ORB, Top 10)



```
[BF] total matches: 111
[BF] matching time: 0.76 ms
```

Matching with RANSAC (ORB, Top 10)



```
[RANSAC] inliers: 26 / 111 (23.4%)
[RANSAC] estimation+filtering time: 10.62 ms
[TOTAL] BF + RANSAC time: 11.38 ms
```

quite expensive
(Random Sampling + Iteration)

Conclusion

- RANSAC is an **iterative random sampling method** for robust model fitting and outlier rejection
  - Lo-RANSAC adds **local optimization**, improving accuracy and convergence
- Complexity (Big O)
  - RANSAC ≈ O(T·N) (T는 반복 횟수)
  - Lo-RANSAC ≈ O(N·logN)
- Applications
  - Computer Vision (Local feature matching, Visual Odometry …)

    ⇒ Slower in local feature matching, but highly effective in eliminating outliers and ensuring geometric consistency

Reference

- https://en.wikipedia.org/wiki/Random_sample_consensus
- https://gaussian37.github.io/vision-concept-ransac/#ransac%EC%9D%98-%EC%9D%B4%EB%9E%80-1
- https://www.youtube.com/watch?v=Q7FqV_bglHo
- https://darkpgmr.tistory.com/61
- https://commons.wikimedia.org/wiki/File:RANSAC_LINIE_Animiert.gif
- https://pmc.ncbi.nlm.nih.gov/articles/PMC9824669/
- https://blog.csdn.net/weixin_42279404/article/details/80715380
- https://medium.com/xulab/review-superglue-learning-feature-matching-with-graph-neural-networks-19d49ca481c4
- https://www.thinkautonomous.ai/blog/visual-slam/
- https://www.geeksforgeeks.org/computer-vision/feature-matching-in-computer-vision-techniques-and-applications/
- https://www.theguardian.com/us-news/2025/may/22/ginger-cats-mystery-solved