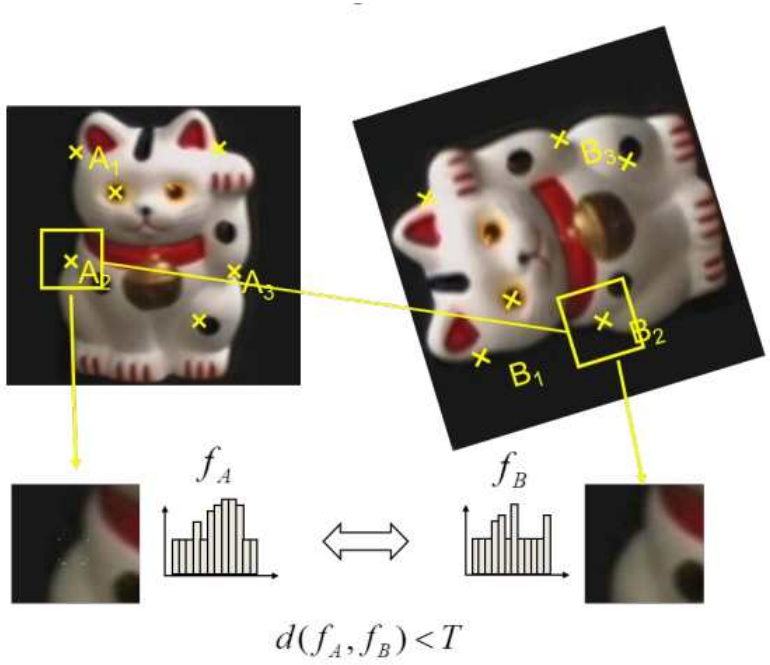
Feature Matching

Feature Matching

- 정의
  - 두 영상에서 추출한 특징점 기술자를 비교하여 서로 비슷한 특징점을 찾는 작업
- Overview



1. Detect a set of distinct feature points
2. Define a patch around each point
3. Extract and normalize the patch
4. Compute a local descriptor
5. Match local descriptors

Feature Matching

- Distance between descriptors

  - L_1 distance (SAD):

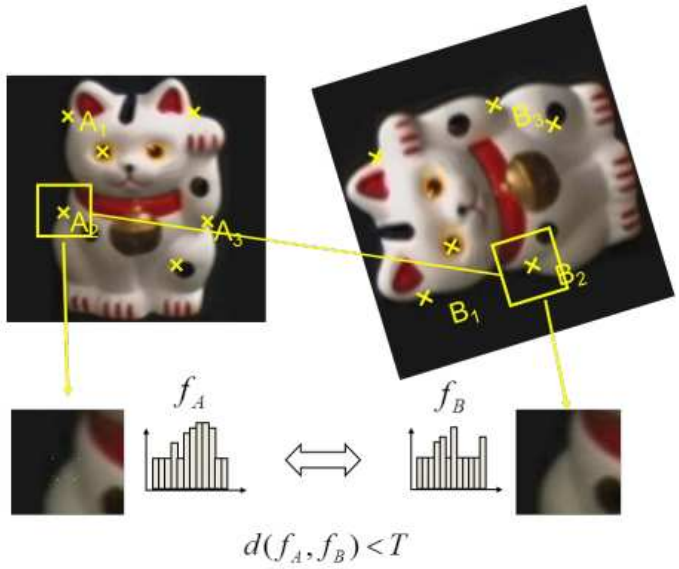
$$d(f_a, f_b) = \sum |f_a - f_b|$$

  - L_2 distance (SSD):

$$d(f_a, f_b) = \sum (f_a - f_b)^2$$

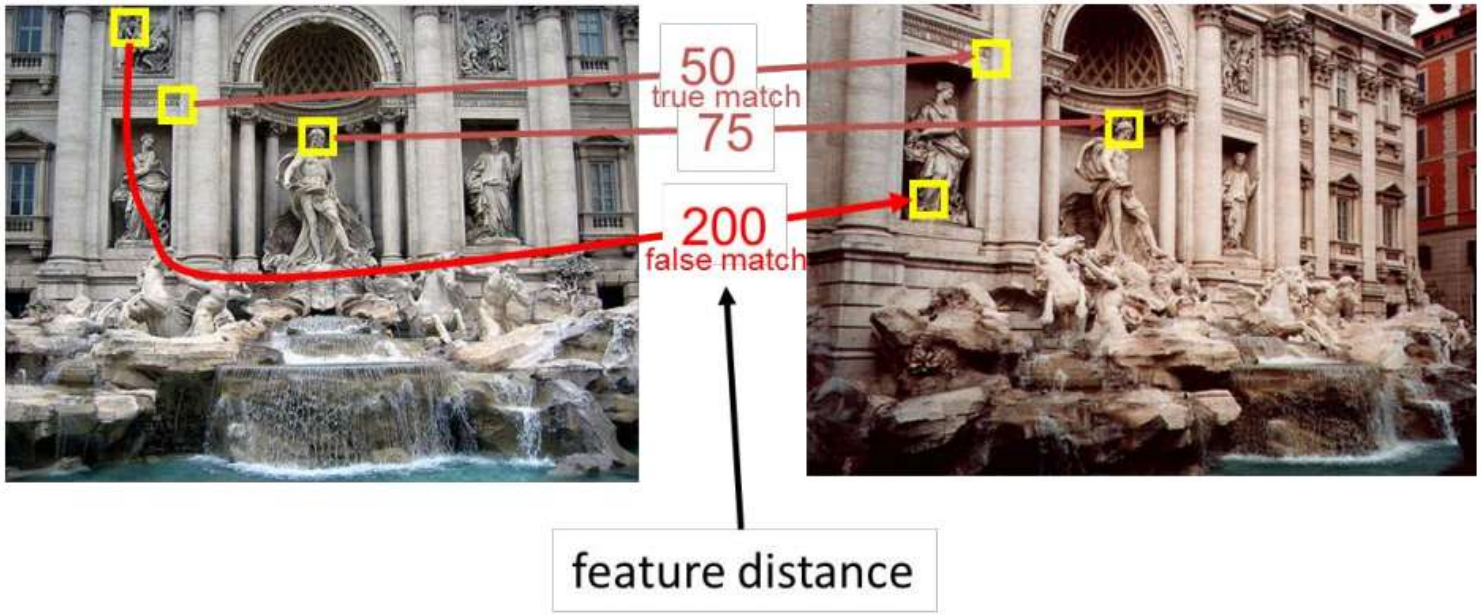
  - Hamming distance:

$$d(f_a, f_b) = \sum \mathrm{XOR}(f_a, f_b)$$



Feature Matching

- Distance Threshold
  - The distance threshold affects performance



Feature Matching

- Matching performance

		실제 정답	
		Positive	Negative
실험 결과	Positive	True Positive	False Positive (Type 1 Error)
	Negative	False Negative (Type 2 Error)	True Negative

$$\text{Precision} = \frac{tp}{tp + fp} \quad \text{(정확률)}$$

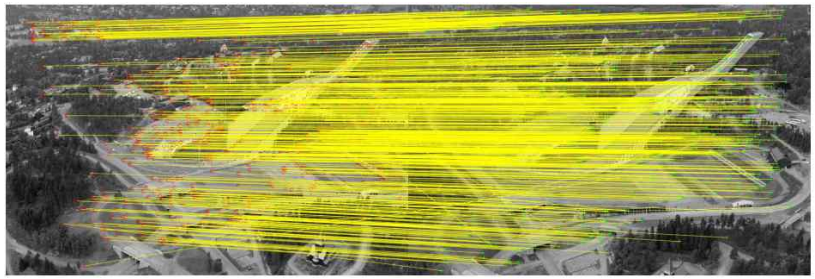
$$\text{Recall} = \frac{tp}{tp + fn} \quad \text{(재현률)}$$

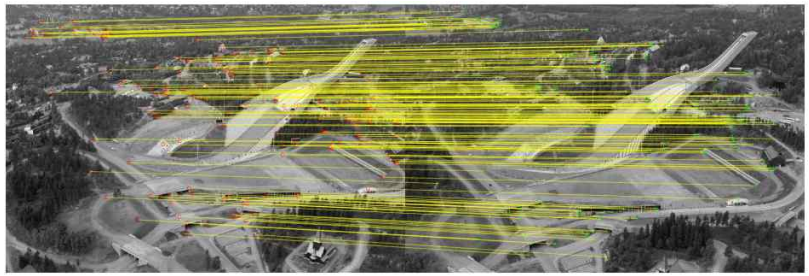
$$F_\beta = (1 + \beta^2)\frac{\text{정확률} \times \text{재현율}}{\beta^2 \times \text{정확률} + \text{재현율}}$$

$$F_1 = \frac{2 \times \text{정확률} \times \text{재현율}}{\text{정확률} + \text{재현율}}$$

Feature Matching

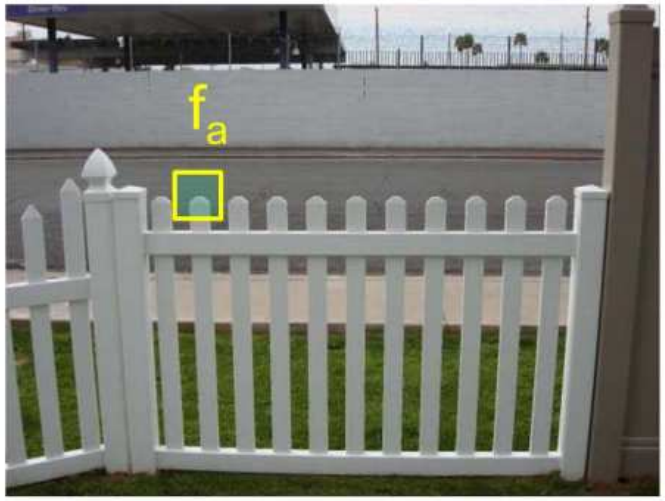
- Matching Strategy
  1) Compare all
  2) Take the closest
     - Or k closest
     - And/Or within a (low) threshold distance
  3) Choose the N best putative matches



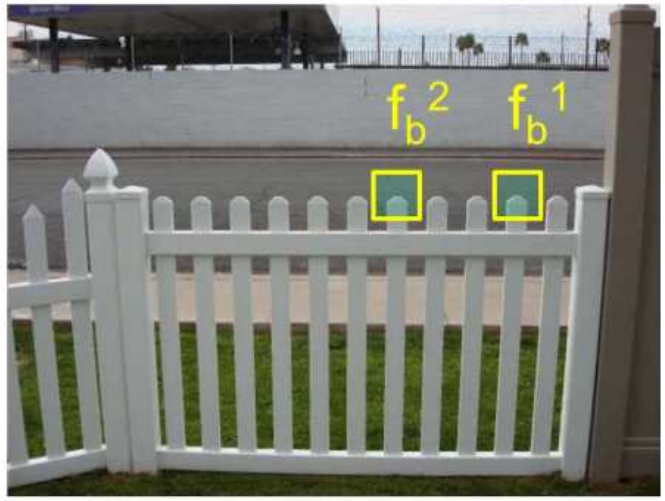


Feature Matching

- Distance ratio
  - For a descriptor f_a in I_a, take the two closest descriptors f_b^1 and f_b^2 in I_b
  - Perform ratio test: $d(f_a, f_b^1) / d(f_a, f_b^2)$
    - Low distance ratio: f_b^1 can be a good match
    - High distance ratio: f_b^1 can be an ambiguous or incorrect match



I_a

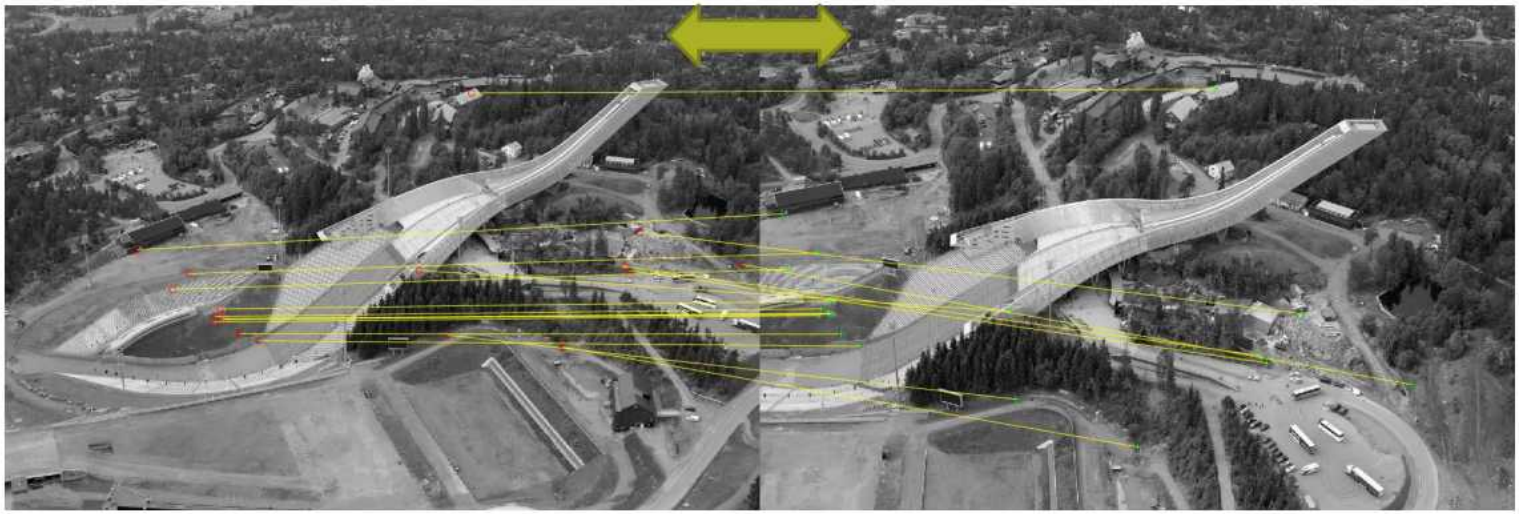


I_b

Feature Matching

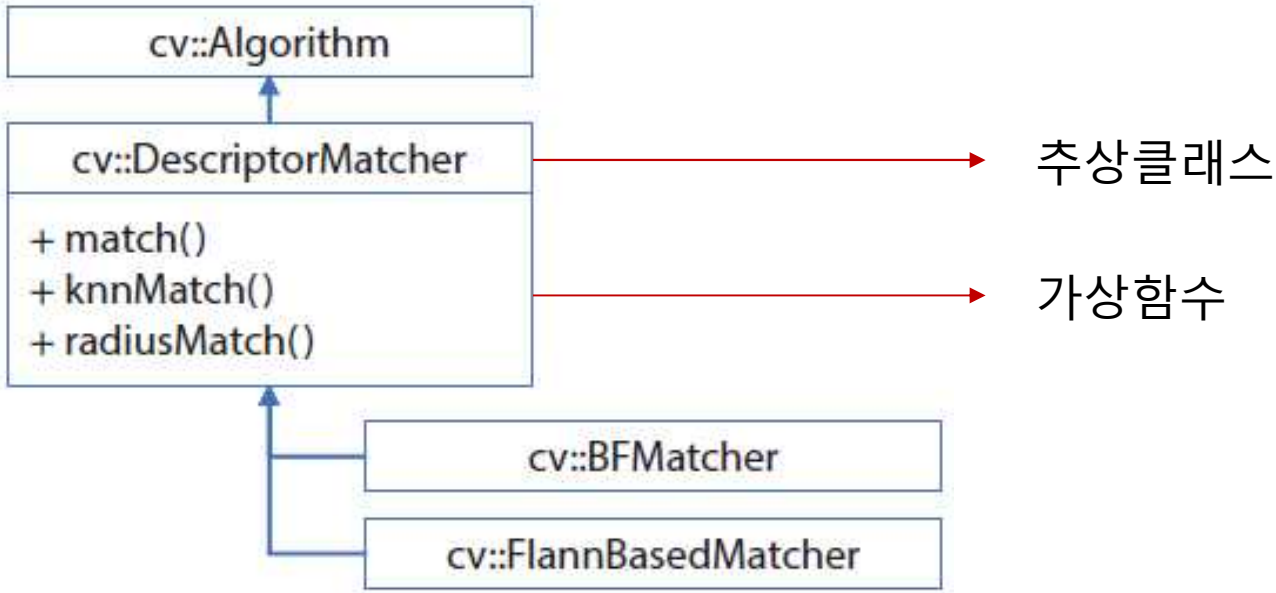
- Cross check test

  - Choose matches (f_a, f_b) so that
    - f_b is the best match for f_a in I_b
    - And f_a is the best match for f_b in I_a
  - Alternative to ratio test



OpenCV

- Feature matching class



- match(): 1개 리턴
- knnMatch(): 최근접 k 개 리턴
- radiusMatch(): 지정된 반경 내 모두 리턴

- BFMatcher: Brute-Force matching (전수조사) (ex: 1000 특징 x 2000 특징 = 2,000,000 번 비교)
- FlannBasedMatcher: Flann (Fast library approximate nearest neighbor) 사용 matching (일부 조사)

OpenCV

- DMatch 클래스
  - 한 장의 영상에서 추출한 특징점과 다른 한 장의 영상, 또는 여러 영상에서 추출한 특징점 사이의 매칭 정보를 표현

코드 14-5 간략화한 DMatch 클래스 정의

```
01  class DMatch
02  {
03  public:
04      DMatch();
05      DMatch(int _queryIdx, int _trainIdx, float _distance);
06      DMatch(int _queryIdx, int _trainIdx, int _imgIdx, float _distance);
07
08      int queryIdx;
09      int trainIdx;
10      int imgIdx;
11
12      float distance;
13
14      bool operator<(const DMatch &m) const;
15  };
```

8행	DMatch::queryIdx 멤버 변수는 질의 기술자 번호를 나타냅니다.
9행	DMatch::trainIdx 멤버 변수는 훈련 기술자 번호를 나타냅니다.
10행	DMatch::imgIdx 멤버 변수는 훈련 영상 번호를 나타냅니다. 여러 장의 영상을 훈련 영상으로 설정한 경우에 사용됩니다.
12행	DMatch::distance 멤버 변수는 두 기술자 사이의 거리를 나타냅니다.
14행	DMatch 클래스에 대한 크기 비교 연산자 재정의이며, DMatch::distance 멤버 변수 값을 이용하여 크기를 비교합니다.

OpenCV

- 객체 생성
  - BFMatcher

```
static Ptr<BFMatcher> BFMatcher::create(int normType = NORM_L2,
                                        bool crossCheck = false);
```

• normType	기술자 거리 측정 방식. NORM_L1, NORM_L2, NORM_HAMMING, NORM_HAMMING2 중 하나를 지정합니다. (SIFT, SURF,KAZE => NORM_L1, NORM L2, ORB/BRIEF/KAZE => NORM_HAMMING)
• crossCheck	이 값이 true이면 i번째 질의 기술자와 가장 유사한 훈련 기술자가 j이고, j번째 훈련 기술자와 가장 유사한 질의 기술자가 i인 경우에만 매칭 결과로 반환합니다.
• 반환값	BFMatcher 객체를 참조하는 Ptr 스마트 포인터 객체

```
Ptr<DescriptorMatcher> matcher = BFMatcher::create();
```

  - FlannBasedMatcher

```
static Ptr<FlannBasedMatcher> FlannBasedMatcher::create();
```

• 반환값	FlannBasedMatcher 객체를 참조하는 Ptr 스마트 포인터 객체

normType: NORM_L2

OpenCV

- Matching

```
void DescriptorMatcher::match(InputArray queryDescriptors,
                              InputArray trainDescriptors,
                              std::vector<DMatch>& matches,
                              InputArray mask = noArray()) const;
```

• queryDescriptors	질의 기술자 집합
• trainDescriptors	훈련 기술자 집합
• matches	매칭 결과
• mask	서로 매칭 가능한 질의 기술자와 훈련 기술자를 지정할 때 사용합니다. 행 개수는 질의 기술자 개수와 같아야 하고, 열 개수는 훈련 기술자 개수와 같아야 합니다. 만약 mask.at<uchar>(i,j) 값이 0이면 queryDescriptors[i]는 trainDescriptors[j]로 매칭될 수 없습니다. noArray()를 지정하면 모든 가능한 매칭을 찾습니다.

```
vector<DMatch> matches;
matcher->match(desc1, desc2, matches);
```

OpenCV

- Matching 그리기

```
void drawMatches(InputArray img1, const std::vector<KeyPoint>& keypoints1,
                 InputArray img2, const std::vector<KeyPoint>& keypoints2,
                 const std::vector<DMatch>& matches1to2,
                 InputOutputArray outImg,
                 const Scalar& matchColor = Scalar::all(-1),
                 const Scalar& singlePointColor = Scalar::all(-1),
                 const std::vector<char>& matchesMask = std::vector<char>(),
                 DrawMatchesFlags flags = DrawMatchesFlags::DEFAULT);
```

• img1	첫 번째 입력 영상
• keypoints1	첫 번째 입력 영상에서 검출된 특징점
• img2	두 번째 입력 영상
• keypoints2	두 번째 입력 영상에서 검출된 특징점
• matches1to2	첫 번째 입력 영상에서 두 번째 입력 영상으로의 매칭 정보
• outImg	출력 영상
• matchColor	매칭된 특징점과 직선 색상. 만약 Scalar::all(-1)을 지정하면 임의의 색상으로 그립니다.
• singlePointColor	매칭되지 않은 특징점 색상. 만약 Scalar::all(-1)을 지정하면 임의의 색상으로 그립니다.
• matchesMask	매칭 정보를 선택하여 그릴 때 사용할 마스크. 만약 std::vector<char>()를 지정하면 모든 매칭 결과를 그립니다.
• flags	매칭 정보 그리기 방법. DrawMatchesFlags 열거형 상수를 지정합니다.

OpenCV

코드 14-6 키포인트 매칭 예제 [ch14/matching]

```
01  void keypoint_matching()
02  {
03      Mat src1 = imread("box.png", IMREAD_GRAYSCALE);
04      Mat src2 = imread("box_in_scene.png", IMREAD_GRAYSCALE);
05
06      if (src1.empty() || src2.empty()) {
07          cerr << "Image load failed!" << endl;
08          return;
09      }
10
11      Ptr<Feature2D> feature = ORB::create();
12
13      vector<KeyPoint> keypoints1, keypoints2;
14      Mat desc1, desc2;
15      feature->detectAndCompute(src1, Mat(), keypoints1, desc1);
16      feature->detectAndCompute(src2, Mat(), keypoints2, desc2);
17
18      Ptr<DescriptorMatcher> matcher = BFMatcher::create(NORM_HAMMING);
19
20      vector<DMatch> matches;
21      matcher->match(desc1, desc2, matches);
22
23      Mat dst;
24      drawMatches(src1, keypoints1, src2, keypoints2, matches, dst);
25
26      imshow("dst", dst);
27
28      waitKey();
29      destroyAllWindows();
30  }
```

OpenCV

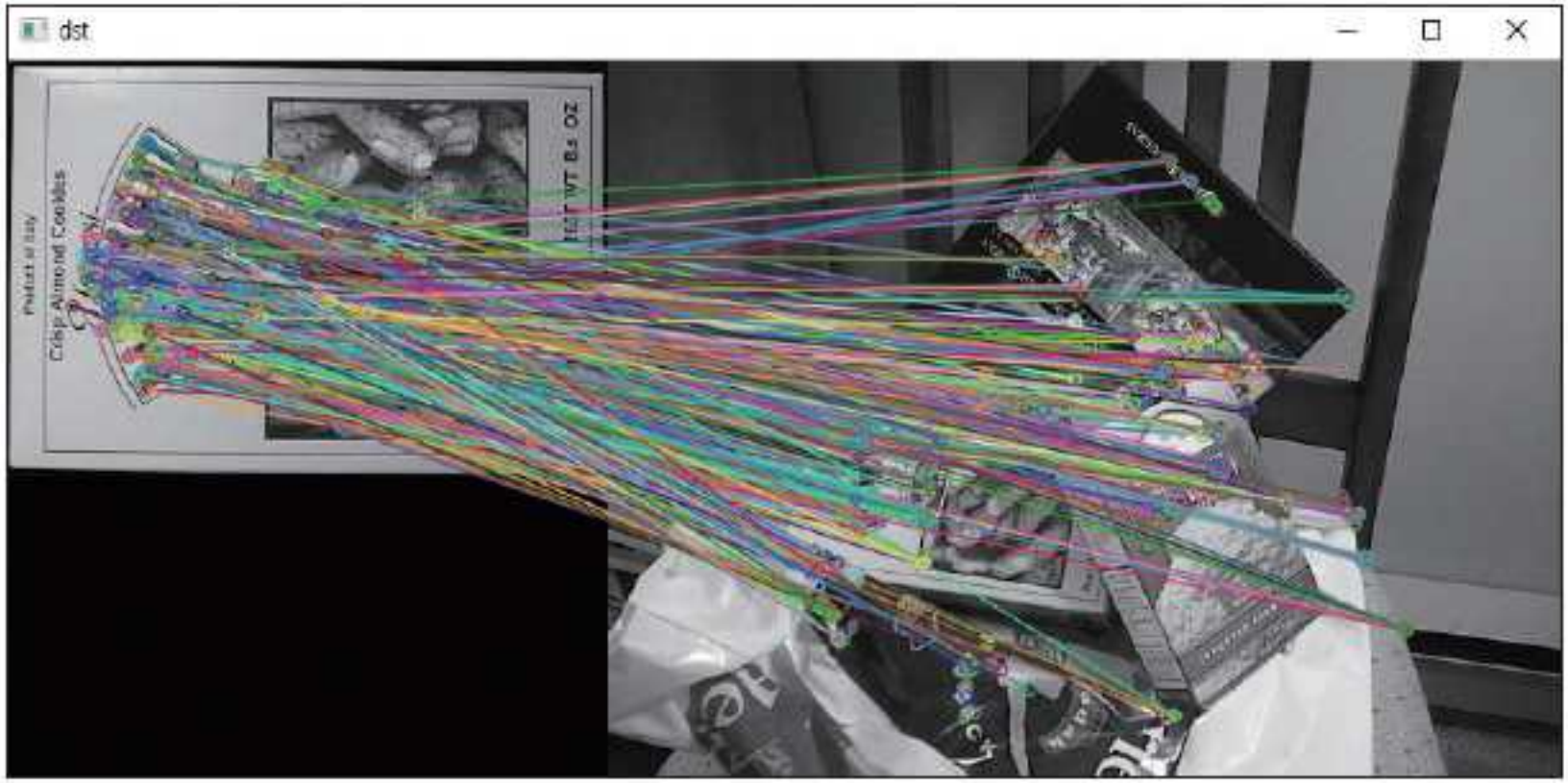
- 입력 영상

(a)

(b)

OpenCV

- 결과 영상



OpenCV

- Matching 선별
  - Distance 값 기준으로 오름차순 정렬 후 일정 개수의 matching 만 추출

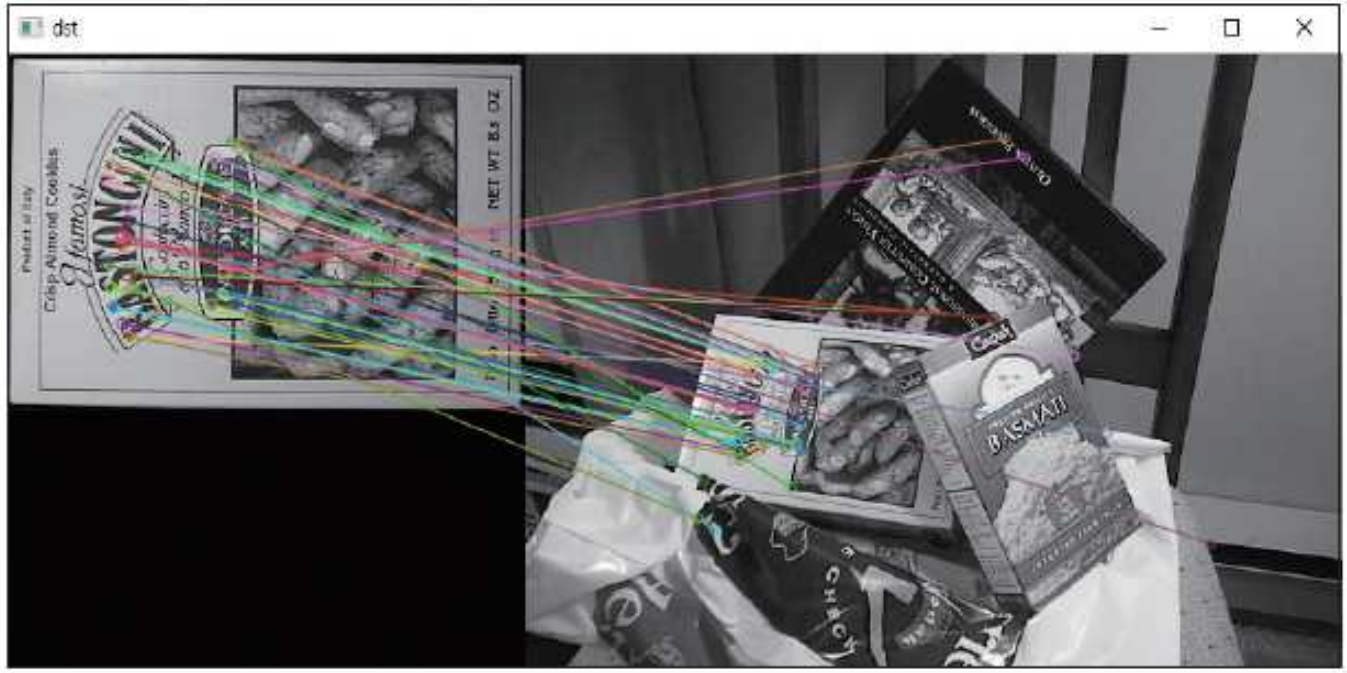
```
vector<DMatch> matches;
matcher->match(desc1, desc2, matches);

std::sort(matches.begin(), matches.end());

vector<DMatch> good_matches(matches.begin(), matches.begin() + 50);
```

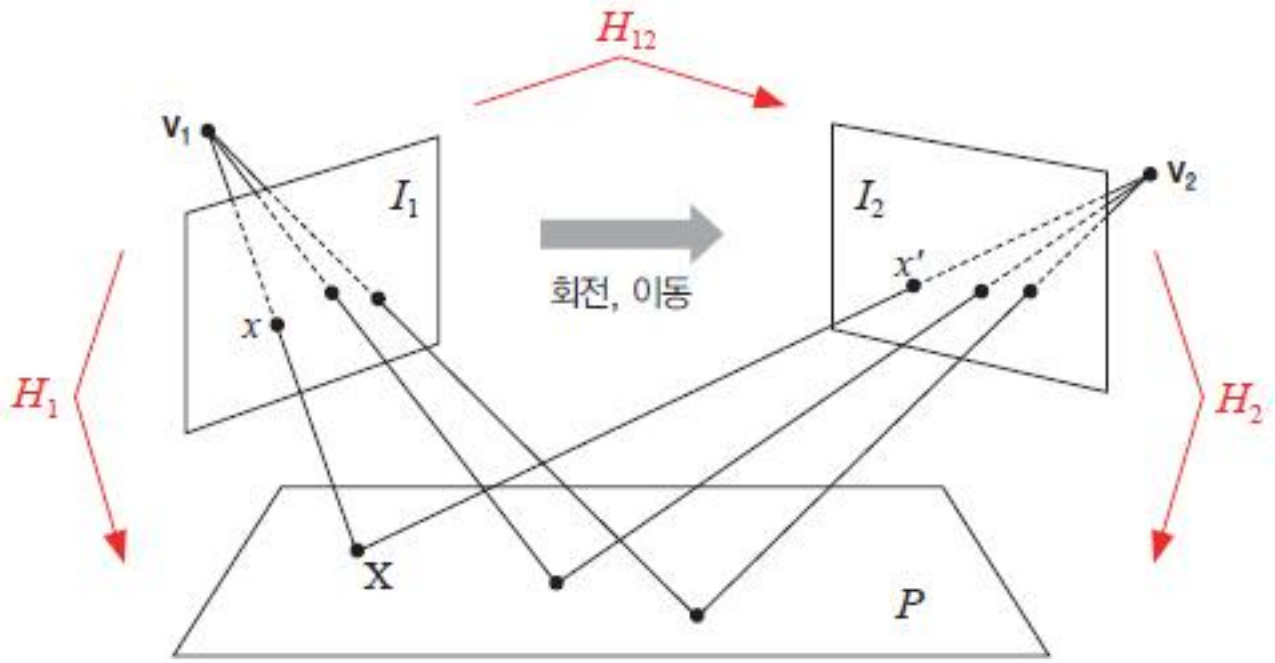
OpenCV

- 결과영상



Homography & Matching

- Homography
  - 3차원 공간상의 평면을 서로 다른 시점에서 바라봤을 때 획득되는 영상 사이의 관계



Homography & Matching

- Perspective Transformation

$$s_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

cf) affine transformation

  - 원본 평면상의 점 (x_i, y_i)
  - 목표 평면상의 점 (x_i', y_i')

feature matching

  - $h_{ij}(1 \le i, j \le 3)$ 는 4개 이상의 (x_i, y_i) - (x_i', y_i') 쌍에 대하여, 다음을 최소화 시키는 값으로 결정

$$\sum_i \left(x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 - \left(y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

Homography & Matching

- OpenCV 함수

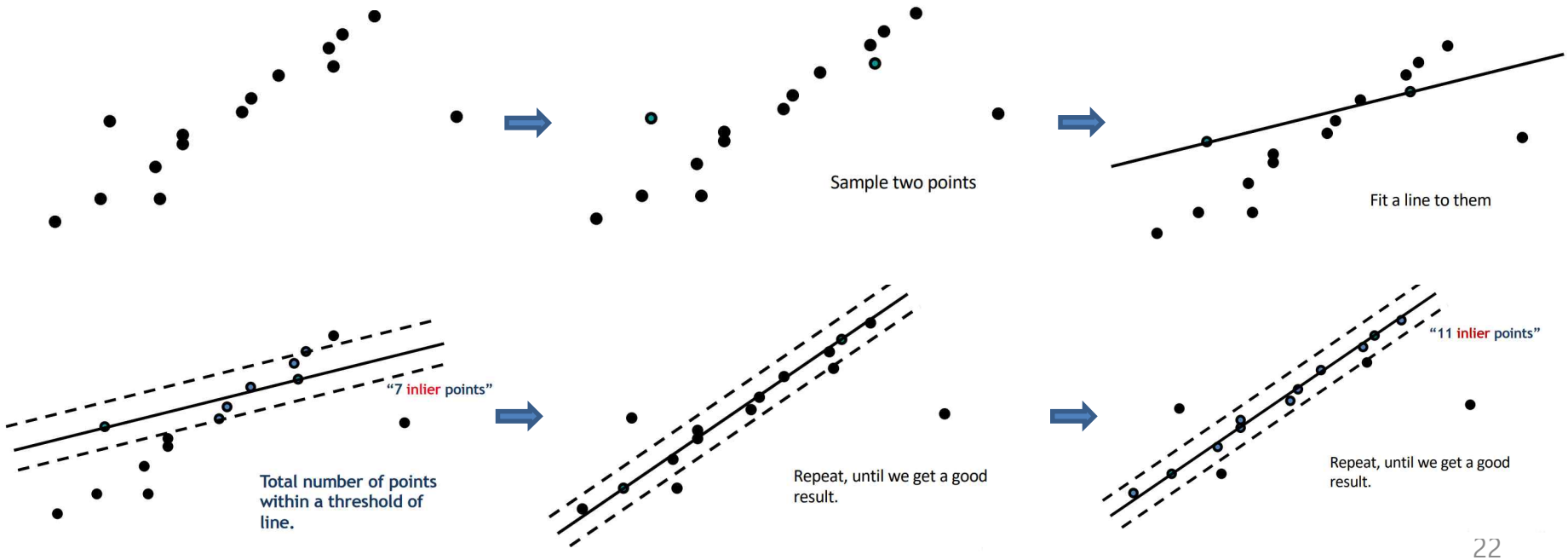
```
Mat findHomography(InputArray srcPoints, InputArray dstPoints,
                   int method = 0, double ransacReprojThreshold = 3,
                   OutputArray mask = noArray(), const int maxIters = 2000,
                   const double confidence = 0.995);
```

• srcPoints	원본 평면상의 점 좌표. CV_32FC2 타입의 Mat 객체 또는 vector<Point2f> 타입의 변수를 지정합니다.
• dstPoints	목표 평면상의 점 좌표. CV_32FC2 타입의 Mat 객체 또는 vector<Point2f> 타입의 변수를 지정합니다.
• method	호모그래피 행렬 계산 방법. 다음 방법 중 하나를 지정합니다. • 0 – 모든 점을 사용하는 일반적인 방법. 최소자승법 • LMEDS – 최소 메디안 제곱(least-median of squares) 방법 • RANSAC – RANSAC 방법 • RHO – PROSAC 방법
• ransacReprojThreshold	최대 허용 재투영 에러. 이 값 이내로 특징점이 재투영되는 경우에만 정상치로 간주합니다. RANSAC과 RHO 방법에서만 사용됩니다.
• mask	호모그래피 계산에 사용된 점들을 알려 주는 출력 마스크 행렬. LMEDS와 RANSAC 방법에서만 사용됩니다.
• maxIters	RANSAC 최대 반복 횟수
• confidence	신뢰도 레벨. 0에서 1 사이의 실수를 지정합니다.
• 반환값	CV_64FC1 타입의 3×3 호모그래피 행렬을 반환합니다. 만약 호모그래피를 계산할 수 없는 상황이라면 비어 있는 Mat 객체가 반환됩니다.

Homography & Matching

- RANSAC 알고리즘
  - Random Sample Consensus [Fitschler & Bolles 1981]

(ex) line fitting example



Homography & Matching

- RANSAC Parameters

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
  n — the smallest number of points required
  k — the number of iterations required
  t — the threshold used to identify a point that fits well
  d — the number of nearby points required
    to assert a model fits well
Until k iterations have occurred
  Draw a sample of n points from the data
    uniformly and at random
  Fit to that set of n points
  For each data point outside the sample
    Test the distance from the point to the line
      against t; if the distance from the point to the line
      is less than t, the point is close
  end
  If there are d or more points close to the line
    then there is a good fit. Refit the line using all
    these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

Homography & Matching

코드 14-8 키포인트 매칭 및 호모그래피 계산 예제 [ch14/matching]

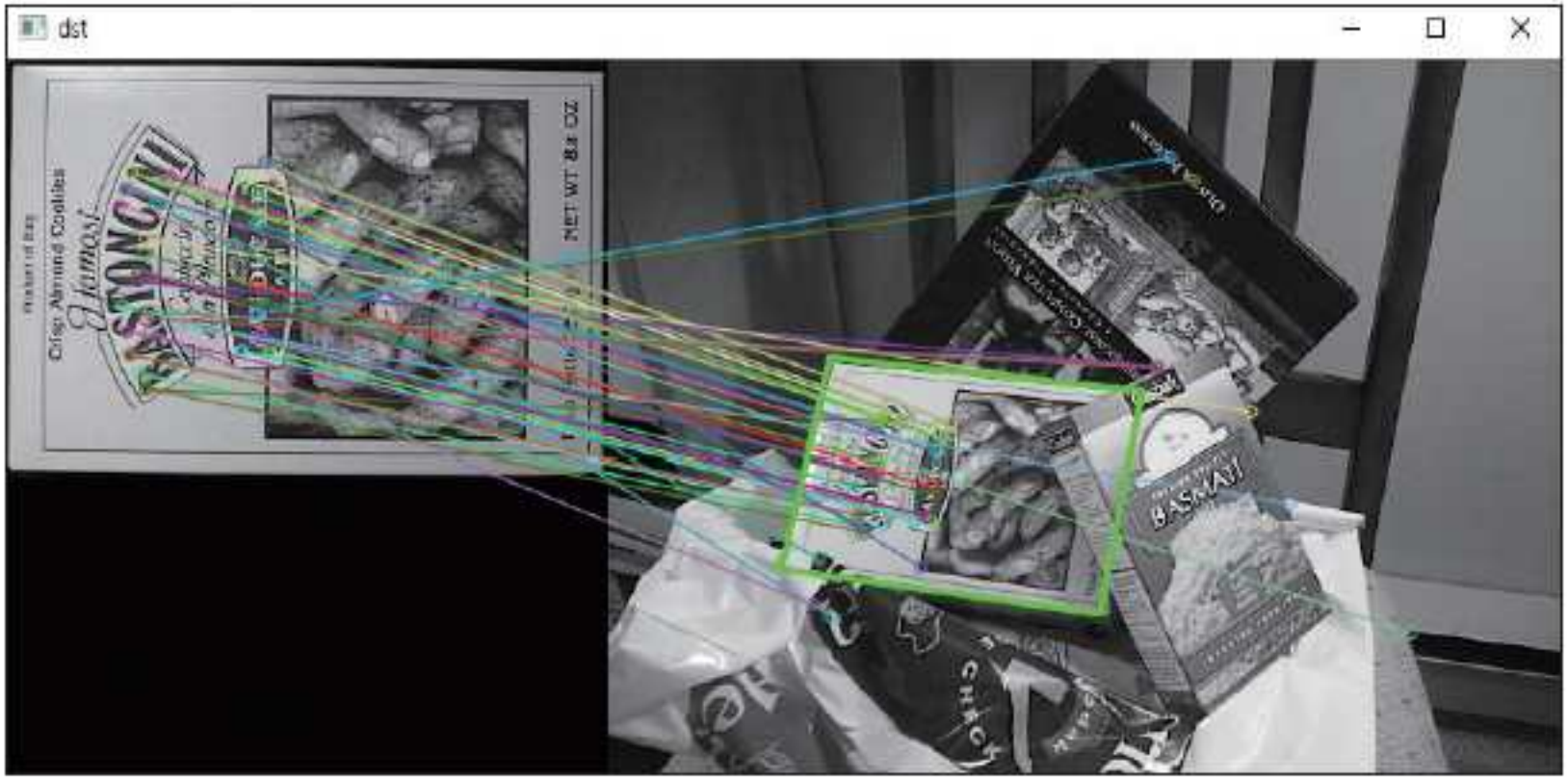
```
01  void find_homography()
02  {
03      Mat src1 = imread("box.png", IMREAD_GRAYSCALE);
04      Mat src2 = imread("box_in_scene.png", IMREAD_GRAYSCALE);
05
06      if (src1.empty() || src2.empty()) {
07          cerr << "Image load failed!" << endl;
08          return;
09      }
10
11      Ptr<Feature2D> orb = ORB::create();
12
13      vector<KeyPoint> keypoints1, keypoints2;
14      Mat desc1, desc2;
15      orb->detectAndCompute(src1, Mat(), keypoints1, desc1);
16      orb->detectAndCompute(src2, Mat(), keypoints2, desc2);
17
18      Ptr<DescriptorMatcher> matcher = BFMatcher::create(NORM_HAMMING);
19
20      vector<DMatch> matches;
21      matcher->match(desc1, desc2, matches);
22
23      std::sort(matches.begin(), matches.end());
24      vector<DMatch> good_matches(matches.begin(), matches.begin() + 50);
25
```

Homography & Matching

```
26      Mat dst;
27      drawMatches(src1, keypoints1, src2, keypoints2, good_matches, dst,
28          Scalar::all(-1), Scalar::all(-1), vector<char>(),
29          DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
30
31      vector<Point2f> pts1, pts2;
32      for (size_t i = 0; i < good_matches.size(); i++) {
33          pts1.push_back(keypoints1[good_matches[i].queryIdx].pt);
34          pts2.push_back(keypoints2[good_matches[i].trainIdx].pt);
35      }
36
37      Mat H = findHomography(pts1, pts2, RANSAC);
38
39      vector<Point2f> corners1, corners2;
40      corners1.push_back(Point2f(0, 0));
41      corners1.push_back(Point2f(src1.cols - 1.f, 0));
42      corners1.push_back(Point2f(src1.cols - 1.f, src1.rows - 1.f));
43      corners1.push_back(Point2f(0, src1.rows - 1.f));
44      perspectiveTransform(corners1, corners2, H);
45
46      vector<Point> corners_dst;
47      for (Point2f pt : corners2) {
48          corners_dst.push_back(Point(cvRound(pt.x + src1.cols), cvRound(pt.y)));
49      }
50
51      polylines(dst, corners_dst, true, Scalar(0, 255, 0), 2, LINE_AA);
52
53      imshow("dst", dst);
54
55      waitKey();
56      destroyAllWindows();
57  }
```

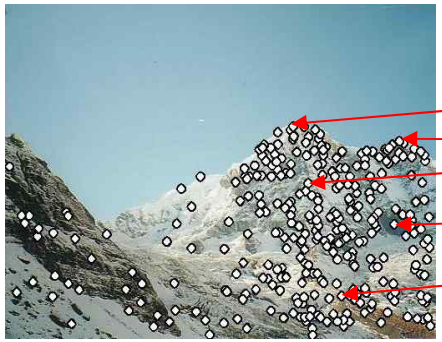
Homography & Matching

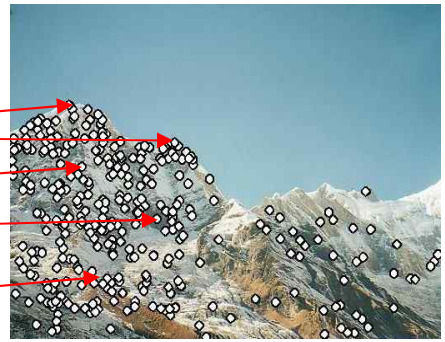
- 실행화면



Building Panorama

- Idea





Feature detection
& Matching

Homography
& Transformation

Building Panorama

- OpenCV 클래스: Stitcher

```
static Ptr<Stitcher> Stitcher::create(Mode mode = Stitcher::PANORAMA);
```

- mode : 이어 붙이기 방식. Stitcher::PANORAMA 또는 Stitcher::SCANS를 지정합니다.
- 반환값 : Stitcher 객체를 참조하는 Ptr 스마트 포인터 객체

```
Ptr<Stitcher> stitcher = Stitcher::create();
```

Building Panorama

코드 14-9 영상 이어 붙이기 예제 프로그램 [ch14/stitching]

```
01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace std;
06
07  int main(int argc, char* argv[])
08  {
09      if (argc < 3) {
10          cerr << "Usage: stitching.exe <image_file1> <image_file2> [<image_file3>
    ...]" << endl;
11          return -1;
12      }
13
14      vector<Mat> imgs;
15      for (int i = 1; i < argc; i++) {
16          Mat img = imread(argv[i]);
17
18          if (img.empty()) {
19              cerr << "Image load failed!" << endl;
20              return -1;
21          }
22
23          imgs.push_back(img);
24      }
25
```

Building Panorama

```
26      Ptr<Stitcher> stitcher = Stitcher::create();
27
28      Mat dst;
29      Stitcher::Status status = stitcher->stitch(imgs, dst);
30
31      if (status != Stitcher::Status::OK) {
32          cerr << "Error on stitching!" << endl;
33          return -1;
34      }
35
36      imwrite("result.jpg", dst);
37
38      imshow("dst", dst);
39
40      waitKey();
41      return 0;
42  }
```

Building Panorama

- 실행화면 `stitching.exe img1.jpg img2.jpg img3.jpg`