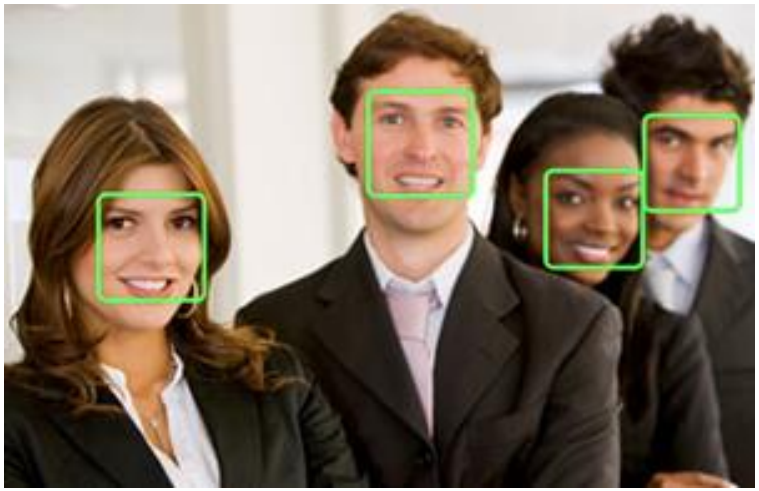
Image Features

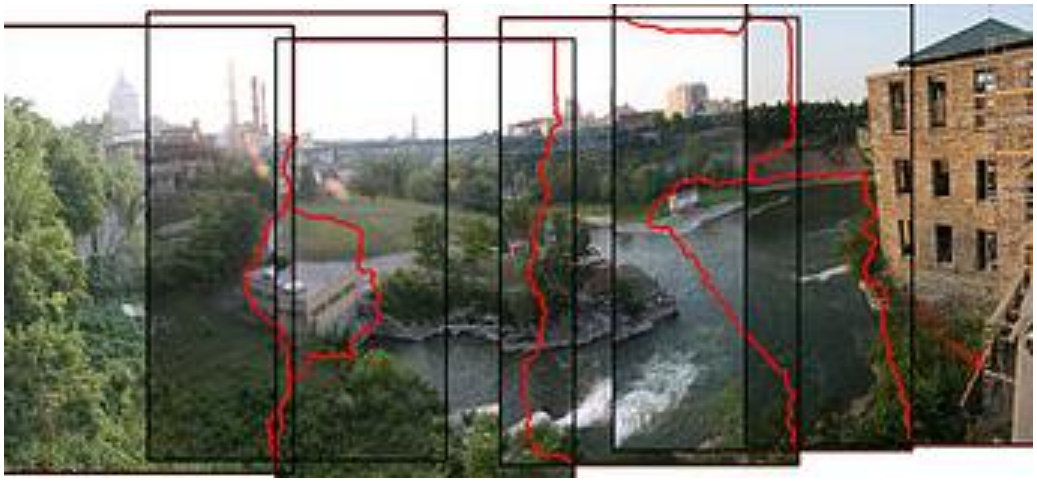
Sung Soo Hwang

What is an image feature?

- An image feature is a piece of information which is relevant for solving the computational task related to a certain application.
- Features may be specific structures in the image such as points, edges or objects.
- Features may also be the result of a general neighborhood operation or feature detection applied to the image.

Why do we need to extract a feature?

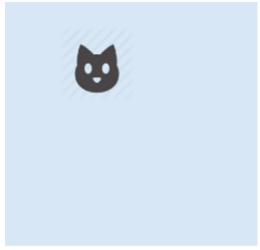
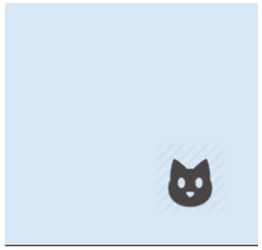


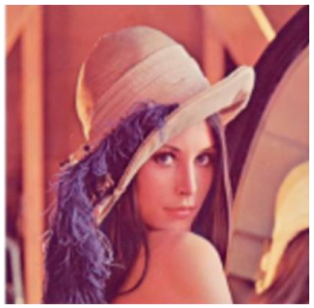
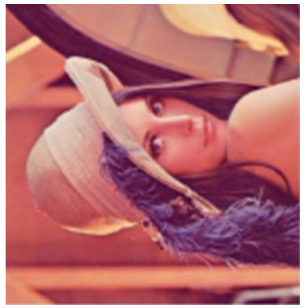
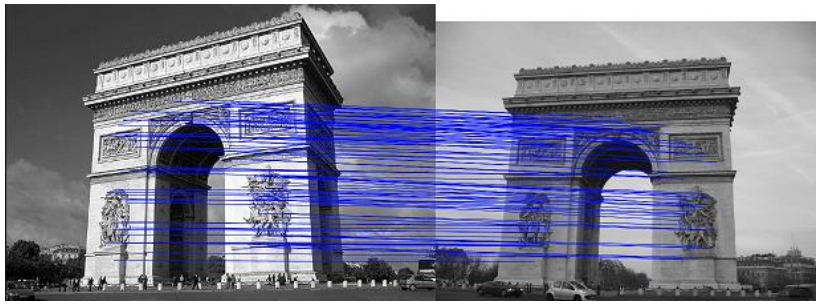


What is a good feature?

- A good feature should be invariant to….
  - Illumination
  - Translation
  - Scale
  - Rotation
  - Perspective transform



https://www.semanticscholar.org/paper/Illumination-Invariant-Imaging-%3A-Applications-in-%2C-Maddern-Stewart/7647e5bc73a61a2a9201e26b682e25e2877f4681

https://stats.stackexchange.com/questions/207195/translational-variance-in-convolutional-neural-networks

http://www.ismailsirma.com/visualsfm-3d-construction-of-images

What is a good feature?

- A good feature should be computationally inexpensive
- A good feature should be memory efficient

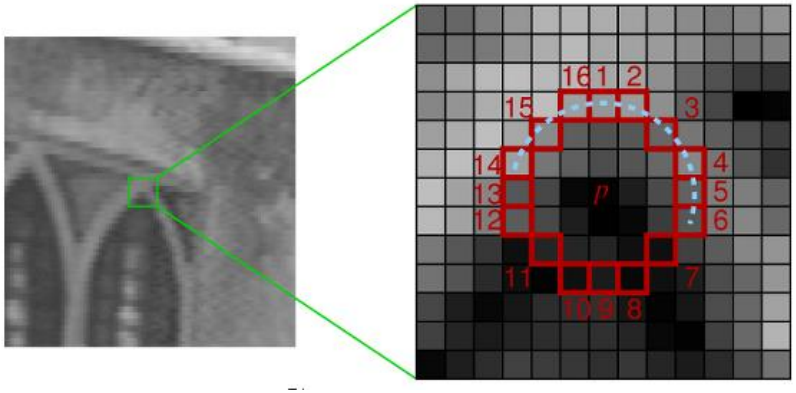
Several Images features

- Widely used feature extractor & descriptor

명칭	detector	descriptor
Harris Corner(1988)	o	x
Shi & Tomasi(1994) (goodFeaturesToTrack)	o	x
SIFT(1999)	o	o
MSER(2004)	o	o
SURF(2006)	o	o
FAST(2006)	o	x
ORB (FAST+BRISK)	o	o
AGAST(2010)	o	o
BRIEF(2012)	o	o
AKAZE(2012) (KAZE)	o	o

ORB

- oFast detector + r-BRIEF descriptor
  - FAST
    - Determines the corner by having more than N consecutive pixels whose intensities are higher(or lower)
    - 9 consecutive pixels when radius is 3



  - BRIEF
    - A bit string descriptor of an image patch constructed from a set of binary intensity tests
  - ORB is fast and illumination/rotation-invariant

Image matching

- Image matching by Feature matching
  - Process
    1. Find features in input images by feature extractor
    2. Describe each feature by feature descriptor
    3. Compare the similarity between features in input images
    4. Extract good matchings
  - What is a good matching?
    - A matching between feature A and B is good when only A and B are similar
    - Good matching can be estimated by Nearest-Neighbor-Distance Ration(NNDR)

$$NNDR(Nearest\ neighbor\ distance\ ratio)$$

$$= \frac{distance\ to\ best\ match}{distance\ to\ second\ best\ match}$$

Image matching

- Result

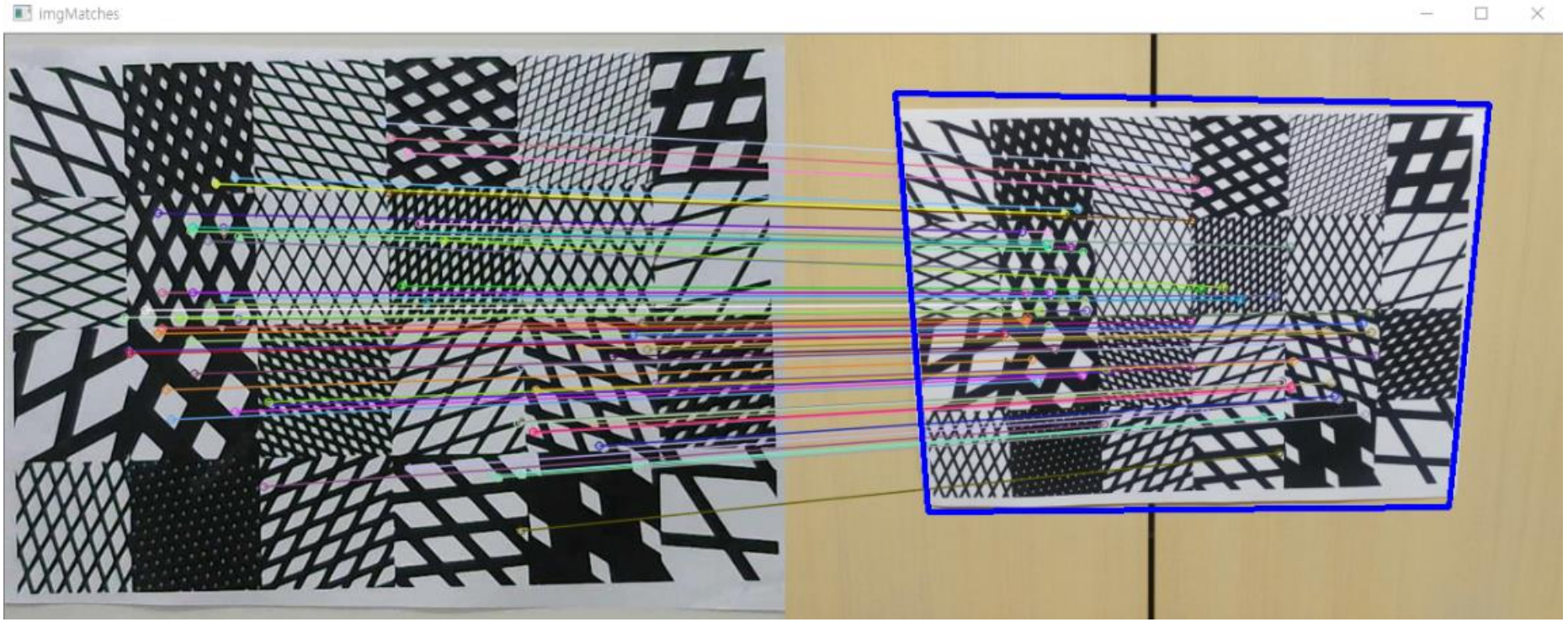


Image matching

- Example code

```
int main(){
	Mat query, image, descriptors1, descriptors2;
	Ptr<ORB> orbF = ORB::create(1000);
	vector<KeyPoint> keypoints1, keypoints2;
	vector< vector< DMatch> > matches;
	vector< DMatch > goodMatches;
	BFMatcher matcher(NORM_HAMMING);
	Mat imgMatches, H;
	vector<Point2f> obj;
	vector<Point2f> scene;
	vector<Point2f> objP(4);
	vector<Point2f> sceneP(4);
	int i, k;
	float nndrRatio;

	query = imread("assets/query.jpg");
	image = imread("assets/input.jpg");

	if (query.empty() || image.empty()) return -1;

	//Compute ORB Features
	resize(image, image, Size(640, 480));
	orbF->detectAndCompute(query, noArray(), keypoints1, descriptors1);
	orbF->detectAndCompute(image, noArray(), keypoints2, descriptors2);
```

Image matching

- Example code

```
//KNN Matching
k = 2;
matcher.knnMatch(descriptors1, descriptors2, matches, k);

nndr = 0.6f;
for (i = 0; i < matches.size(); i++) {
        if (matches.at(i).size() == 2
                        && matches.at(i).at(0).distance
                        <= nndr * matches.at(i).at(1).distance) {
                        goodMatches.push_back(matches[i][0]);
        }
}
//Draw matching
drawMatches(query, keypoints1, image, keypoints2, goodMatches, imgMatches,
Scalar::all(-1), Scalar(-1), vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

if (goodMatches.size() < 4) { cout << "Matching failed" << endl; return 0; }

//Find perspective transform
for (i = 0; i < goodMatches.size(); i++) {
        obj.push_back(keypoints1[goodMatches[i].queryIdx].pt);
        scene.push_back(keypoints2[goodMatches[i].trainIdx].pt);
}
```

Image matching

- Example code

```
        H = findHomography(obj, scene, RANSAC);
        objP[0] = Point2f(0, 0);
        objP[1] = Point2f(query.cols, 0);
        objP[2] = Point2f(query.cols, query.rows);
        objP[3] = Point2f(0, query.rows);

        perspectiveTransform(objP, sceneP, H);

        for (i = 0; i < 4; i++) sceneP[i] += Point2f(query.cols, 0);
        for (i = 0; i < 4; i++)line(imgMatches, sceneP[i], sceneP[(i + 1) % 4], Scalar(255, 0, 0), 4);

        imshow("imgMatches", imgMatches);
}
```