1. 실습 개요

이 실습의 목표는 강의자료의 Feature Detection 실습 #1을 기반으로, 기존 SURF 기반 참고 코드를 SIFT 기반으로 수정하고 단일 이미지에서 특징점을 검출하는 프로그램을 작성하는 것이다. 이후 강의자료 28, 29페이지에서 제시된 요구사항을 반영하여

- SIFT 파라미터를 조정해 feature point의 개수를 늘려 보고, 그때 나타나는 현상을 분석하고
- 영상을 M x N 블록으로 나누어 블록별로 독립적으로 특징점을 추출하는 방법을 구현한 뒤, 장단점을 정리한다.

2. SIFT 기반 Feature Detection 구현 요약

2.1 SURF → SIFT 변경

참고 코드는 원래 SURF(Scale-Invariant Feature Transform)을 이용하여 keypoint와 descriptor를 추출하는 구조였다. 본 실습에서는 특허 만료 이후 OpenCV 기본 모듈에 포함된 SIFT(Scale-Invariant Feature Transform)를 사용하도록 수정하였다.

OpenCV에서 SIFT는 다음과 같이 생성한다.

```
Ptr<SIFT> sift = SIFT::create(
    nfeatures,       // 최대 특징점 개수 (0이면 제한 없음)
    nOctaveLayers,   // 옥타브당 레이어 개수
    contrastThreshold,
    edgeThreshold,
    sigma
);
```

기존 SURF 코드에서 SURF::create()를 호출하던 부분을 위와 같은 SIFT::create()로 대체하고, detect() 혹은 detectAndCompute() 호출 부분은 인터페이스가 거의 동일하므로 최소한의 수정으로 대체할 수 있다.

2.2 단일 영상에서 SIFT keypoint 시각화

main() 함수에서는 다음과 같은 순서로 처리한다.

1. imread()로 컬러 영상을 입력받는다.
2. cvtColor()를 이용하여 BGR 영상을 그레이스케일 영상으로 변환한다.
3. detectSIFTGlobal() 함수를 호출하여 전체 영상에서 SIFT로 keypoint와 descriptor를 추출한다.
4. drawKeypoints()를 사용해서 원본 컬러 영상 위에 keypoint를 원 형태로 시각화한다.
5. 결과 이미지를 창으로 보여주고(sift_global.png) 파일로 저장한다.

이 과정이 강의자료 실습 #1의 "예측된 feature point(keypoints)를 영상에 출력"하는 요구사항에 해당한다.

3. 28페이지: Feature point 개수 증가 실험 및 분석

3.1 SIFT 파라미터와 feature 개수의 관계

feature point 개수는 주로 다음 파라미터에 의해 결정된다.

- **nfeatures** SIFT가 최종적으로 유지할 수 있는 최대 keypoint 개수이다. 0이면 제한이 없고 충분한 특징점이 있으면 많이 뽑힌다.
- **contrastThreshold** DoG(Difference of Gaussian) 응답의 contrast가 이 값보다 작은 점들은 버린다. 값을 작게 줄수록 대비가 약한 영역에서도 특징점을 받아들이므로 특징점 개수가 증가하는 경향이 있다.
- **edgeThreshold** 강한 edge지만 코너가 아닌 점들을 제거하기 위한 임계값이다. 값을 크게 주면 edge 성분을 더 많이 제거하고, 작게 주면 edge 근처에서도 특징점이 많이 남는다.

- **sigma** 최초 옥타브에서 사용되는 Gaussian blur의 표준편차이다. 작은 sigma는 더 세밀한 구조를 남기고, 큰 sigma는 부드럽게 만들어 미세한 특징을 줄이는 방향으로 작용한다.

코드에서는 contrastThreshold를 0.02로 설정하여(OpenCV 기본값 0.04보다 낮음) 상대적으로 많은 feature point가 검출되도록 했다.

3.2 Feature point를 많이 얻기 위한 파라미터 튜닝 전략
Feature point의 개수를 늘리고 싶을 때 기본적인 전략은 다음과 같다.

1. **contrastThreshold를 낮춘다.**
    - 대비가 작은 영역에서도 keypoint를 받아들이므로 전체 개수가 크게 증가한다.
    - 단점은 노이즈나 중요하지 않은 세부 구조까지 특징점으로 선택될 수 있다는 점이다.
2. **nfeatures를 증가시킨다.**
    - SIFT 내부에서 응답이 큰 순서대로 상위 nfeatures 개만 남기므로 nfeatures를 줄이면 중요한 점만 남고, 늘리면 더 많은 점을 유지하게 된다.
3. **edgeThreshold를 적절히 조절한다.**
    - edgeThreshold를 줄이면 edge 근처에서도 더 많은 keypoint가 남지만, edge에만 특징점이 몰리는 현상이 심해질 수 있다.

실습에서 파라미터를 여러 조합으로 바꿔가며 실행하면

- contrastThreshold를 낮추고 nfeatures를 크게 줄수록 전체 keypoint 수가 급격히 증가하고
- 낮은 대비 영역 배경 텍스처 노이즈까지 feature로 잡히는 경향이 생기며
- descriptor 계산 및 이후 매칭 연산량이 증가해 처리 시간이 길어지는
  경향을 확인할 수 있다.

3.3 Feature point 개수 증가 시 나타나는 일반적인 현상 분석
파라미터를 조절하여 feature point를 많이 얻으면 일반적으로 다음과 같은 현상이 나타난다.

- **장점**
    - 이미지 전체를 더 촘촘히 샘플링하므로, 객체가 부분적으로 가려지거나 조명이 변하는 경우에도 대응할 수 있는 redundancy가 늘어난다.
    - 이후 매칭 단계에서 사용할 수 있는 후보 점이 많아져 강인한 매칭이 가능해진다.
- **단점**
    - 연산량이 증가한다. keypoint의 수에 비례하여 descriptor 계산 비용과 매칭 비용이 늘어난다.
    - 서로 매우 비슷한 위치에 중복된 keypoint가 많이 생기면, 실제로 유용한 정보량에 비해 descriptor의 수만 늘어나 효율이 떨어진다.
    - 텍스처가 거의 없는 영역에서도 노이즈 기반의 특징점이 생성되어, 매칭에서 오검출(false match)을 늘릴 수 있다.

따라서 28페이지의 요구사항처럼 단순히 feature point를 많이 만드는 것뿐 아니라, 연산량과 특징점 품질 사이의 trade-off를 고려하여 적절한 파라미터를 선택해야 한다.

4. 29페이지: M x N 블록 기반 Feature Point 검출 및 분석
4.1 Feature Point 집중 현상 문제
강의자료 27페이지 예시처럼, 한 장의 이미지 안에서도 텍스처가 풍부한 영역과 단순한 영역이 공존한다. SIFT와 같은 detector를 전체 이미지에 한 번만 적용하면

- 텍스처가 강한 영역(나무, 건물, 질감이 많은 영역)에 특징점이 몰리고

- 평탄한 영역(하늘, 바닥, 단순한 벽)에서는 특징점이 거의 나오지 않는 현상이 나타난다.

이렇게 특정 영역에 feature가 몰리면

- 매칭이나 추적에서 화면의 일부 정보에만 의존하게 되어 전체 구조를 반영하기 어렵고
- 한 영역에서 너무 많은 descriptor를 계산해 연산량이 비효율적으로 커지는 성능 문제가 발생할 수 있다.

4.2 M x N 블록 기반 검출 알고리즘

이 문제를 줄이기 위해 강의자료 29페이지에서는 영상을 M x N 블록으로 나누고 블록별로 독립적으로 특징점을 추출하는 방법을 제안한다. 구현 알고리즘은 다음과 같다.

1. 전체 이미지를 GRID_ROWS x GRID_COLS 개의 블록으로 균일하게 분할한다.
2. 각 블록을 ROI(Rect)를 이용해 잘라낸 뒤, 해당 블록에 대해 SIFT를 수행한다.
3. 블록당 최대 nFeaturesPerBlock 개의 keypoint만 유지하도록 SIFT::create()의 nfeatures 인자를 설정한다.
4. 블록 좌상단 좌표를 기준으로 각 local keypoint의 좌표를 전역 이미지 좌표계로 보정한다.
5. 모든 블록에서 얻은 keypoint를 하나의 벡터로 합친다.

코드의 detectSIFTGrid() 함수가 위 과정을 구현한 것으로, GRID_ROWS와 GRID_COLS를 조정하여 블록 크기를 바꿀 수 있다.

4.3 블록 기반 검출의 장점

- **공간적으로 고르게 분포된 특징점 확보**
  - 각 블록에서 일정 개수의 keypoint를 뽑기 때문에, 이미지의 어느 한 부분에 feature가 과도하게 몰리지 않는다.
  - 영상 전체 영역에 걸쳐 고르게 특징점이 분포하므로, 전체 구조를 반영한 매칭이나 정합(Registration)에 유리하다.
- **연산량 제어 용이**
  - GRID_TOTAL_FEATURES와 GRID_ROWS, GRID_COLS를 조절하여 전체 feature 수를 대략적으로 제어할 수 있다.
  - 특정 블록에서 너무 많은 특징점이 나오더라도 nfeaturesPerBlock으로 상한을 두기 때문에 전체 연산량이 예측 가능하다.
- **물체 단위 인식에 유리**
  - 화면의 여러 위치에 있는 물체 각각에 대해 최소한의 특징점을 확보할 수 있어, 장면 내 여러 물체를 동시에 추적하거나 인식할 때 도움이 된다.

4.4 블록 기반 검출의 단점

- **블록 경계 효과**
  - keypoint 주변 정보가 블록 경계에서 잘리는 경우, 실제로는 좋은 특징점이더라도 해당 블록 안에서 충분한 문맥을 확보하지 못해 검출이 어렵다.
  - 블록 경계 근처에서 서로 다른 블록이 중복된 비슷한 특징점을 생성할 수 있다.
- **텍스처가 거의 없는 블록의 문제**
  - 어떤 블록에는 유의미한 특징이 거의 없을 수 있다. 이 경우에도 nfeaturesPerBlock만큼 keypoint를 강제로 뽑으려고 하면, 노이즈성 특징점이 많아질 수 있다.

- **파라미터 선택의 어려움**
    - GRID_ROWS, GRID_COLS를 어떻게 정할지에 따라 결과가 크게 달라진다. 너무 작은 블록을 사용하면 지역 정보만 보고 전역 구조를 놓칠 수 있고, 너무 큰 블록을 사용하면 원래 문제(특정 영역 집중)가 다시 나타난다.
- **계산량 증가 가능성**
    - 이론적으로는 전체 feature 수에 상한을 두지만, 구현상 detector를 여러 번 호출해야 하므로 detector 초기화 및 ROI 처리 과정에서 약간의 오버헤드가 발생한다.

5. 결론

본 실습에서는 강의자료 Feature Detection 실습 #1을 기반으로, SIFT를 이용하여 단일 이미지에서 feature point를 검출하고 시각화하는 코드를 작성하였다. 이어서 28페이지 요구사항에 따라 SIFT의 contrastThreshold, nfeatures 등의 파라미터를 조절하여 feature point 개수를 증가시키는 방법과 그에 따른 장단점을 분석하였다.

또한 29페이지 요구사항을 반영하여 이미지를 M x N 블록으로 분할하고 블록별로 독립적으로 특징점을 추출하는 알고리즘을 구현하였다. 이를 통해 특정 영역에 feature가 몰리는 문제를 완화하고, 공간적으로 균일한 분포의 특징점을 얻을 수 있음을 확인하였다. 반면 블록 경계 효과나 텍스처가 거의 없는 블록에서의 노이즈성 특징점 증가 등 부작용도 존재함을 알 수 있었다.

실제 응용에서는 영상의 특성과 계산 자원을 고려하여

- 전역 검출과 블록 기반 검출을 적절히 조합하고
- SIFT 파라미터(nfeatures, contrastThreshold, edgeThreshold 등)를 실험적으로 조절하면서

feature point의 품질과 개수 사이의 균형을 맞추는 것이 중요하다.