OpenCV 보고서

19기 예비단원 이원준

1. 기본 개념 정의

- GrayScale : 각 픽셀마다 0~255 사이에 값을 가져 흰색은 255
  검은색은 0을 나타낸다.

- BInary: 이미지를 이진화 시킨 것

- RGB: 각 픽셀마다 RGB 3개의 데이터가 들어가서 컬러 표현가능

- HSV : Hue 색상 (0 ~ 180) , Saturation 채도 (0 ~ 255), Value 명도 (0~255)

HSV는 RGB와 호완 가능한 값으로 이미지의 색상의 구간을 지정하거나 색을 판단할때 장점이 있다.

2. Resize 함수

- 일반적으로 컴퓨터 비전에서 카메라에 찍히는 이미지 그대로 사용하는 경우는 잘 없다.

- 그렇기에 리소스 및 연산량을 고려하여 적절히 이미지 사이즈를 재지정한다.

```
Mat img = imread("Lenna.png");
img_small;

resize(img, img_small, Size(200, 200));
```

- 위 코드 처럼 배열로 저장된 이미지를 img_small에 사이즈 재지정하여 복사한다 .

3. 노이즈 처리

- 가우시안 블러를 활용하면 이미지의 노이즈를 전체적으로 안정화 시킬 수 있다.

- Errode와 dilate를 활용하면 특정 이미지를 부각시키거나 배제할 수 있다.

- 다음 예제 코드를 통해 함수의 활용에 대해 살펴 보자.

```cpp
#include <opencv2/opencv.hpp>
#include <stdio.h>

int main() {
    // 이미지 파일 읽기 (흑백으로 로드)
    cv::Mat image = cv::imread("example.jpg", cv::IMREAD_GRAYSCALE);
    if (image.empty()) {
        printf("이미지를 불러올 수 없습니다.\n");
        return -1;
    }

    // 가우시안 블러 적용 (노이즈 제거)
    cv::Mat gaussian_blur;
    cv::GaussianBlur(image, gaussian_blur, cv::Size(5, 5), 0);

    // 커널 정의 (침식과 확장을 위해 사용)
    cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(5, 5));

    // 침식(Erosion) 적용 (이미지에서 작은 영역 제거)
    cv::Mat erosion;
    cv::erode(gaussian_blur, erosion, kernel);

    // 확장(Dilation) 적용 (작은 구멍 채우기)
    cv::Mat dilation;
    cv::dilate(gaussian_blur, dilation, kernel);

    // 결과 이미지 표시
    cv::imshow("Original Image", image);
    cv::imshow("Gaussian Blur", gaussian_blur);
    cv::imshow("Erosion", erosion);
    cv::imshow("Dilation", dilation);

    // 키 입력 대기
    cv::waitKey(0);
    cv::destroyAllWindows();

    return 0;
}
```

- 해당 코드에 대해서는 아래에서 자세히 설명하겠다.

3-1 . cv::GaussianBlur()의 이해

- 해당 코드를 살펴 보면 이미지가 정상적으로 로드되면 이미지에 cv::GaussianBlur() 함수로 이미지에 가우시안 블러를 적용하여 노이즈를 제거한다.

이때 cv::Size(5, 5)는 커널 크기이며, 0은 가우시안 커널의 표준 편차이다.

- 가우시안 커널에 대해 이해하려면 가우시안 함수에 대해 이해해야한다.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- 위 수식은 가우시안 함수이고 이때 **σ는** 가우시안 분포의 표준편차로, 값이 클수록 블러가 강하게 적용된다. 즉 예제코드의 세번째 파라메터를 의미한다.

- 이 수식의 매개변수인 **x와** *y는* 커널의 위치를 나타내는 좌표로, 커널의 중심을 기준으로 한 상대적 위치이다.

- 커널은 이미지의 각 픽셀이 서로 결합하는 범위로 이해하면되는데 일반적으로 3×3 또는 *5x5* 크기의 커널이 자주 사용된다.

3-2. Erosion과 Dilate 함수의 이해

- 위 예제코드를 이어서 설명하겠다.

- 가우시안 블러가 적용된 이미지는 gaussian_blur 행렬에 저장되고 침식과 확장에 사용되는 커널을 정의한다

- Erosion과 Dilate 함수는 3번째와 4번째 파라메터에 커널 외에도 횟수를 지정할 수 있다. 예제 코드에는 횟수가 지정되지 않아서 기본값인 1회를 기반으로 진행

한다 .

```
te(gaussian_blur, dilation, kernel, cv::Point(-1, -1), iterations);
e(gaussian_blur, erosion, kernel, cv::Point(-1, -1), iterations);
```

- 또한 위 처럼 **cv::Point(-1, -1)을 활용해서** 이 매개변수는 커널의 고정점를 설정하며, (-1, -1)은 커널의 중심을 기준으로 연산할 수 있다.

4. 관심영역 추출

주요 개념

- ROI : 이미지의 일부분으로 이미지에 맞게 다양한 형태로 설정한다. 예를 들어 차선인식시에는 도로부분만 ROI로 설정한다. Region of intereset의 약자이다.

- ROI를 사용하면 컴퓨터 리소스 측면에서 불필요한 연산을 줄이거나 메모리 측면에서 장점이 있다. 또한 카메라의 인식 정확도를 상승시킬 수 있는 장점이 있다.

- 마스크 : 마스크는 이미지 처리나 객체 탐지, 분할 등에서 **관심 있는 특정 영역을 선택하거나 강조**할 때 사용하는 이진 이미지이다. 마스크는 원본 이미지와 같은 크기를 가지며, 각 픽셀이 0 또는 1(흑백) 또는 0 또는 255의 값을 갖습니다. **0**은 비활성화된 영역, **255**는 활성화된 영역을 나타냅니다. 이때 검은색이 진리값 0을 갖는ㄷ다.

- 이미지 행렬 사이의 비트연산에 활용될 수 있는데 예를 들어 bitwise_and 함수를 활용해서 원본이미지와 마스크 이미지를 결합하면 마스크에 활성화된 영역만 남

길 수 있다.

- ROI설정은 아래 예제 코드를 기반으로 설명하겠다.

```cpp
#include <opencv2/opencv.hpp>
#include <vector>

int main() {
    // 이미지 로드
    cv::Mat image = cv::imread("example.jpg");
    if (image.empty()) {
        printf("이미지를 불러올 수 없습니다.\n");
        return -1;
    }

    // ROI 좌표 설정 (사다리꼴 모양)
    int width = image.cols;
    int height = image.rows;
    std::vector<cv::Point> trapezoid_points = {
        cv::Point(width * 0.2, height * 0.9),   // 좌하단
        cv::Point(width * 0.8, height * 0.9),   // 우하단
        cv::Point(width * 0.6, height * 0.6),   // 우상단
        cv::Point(width * 0.4, height * 0.6)    // 좌상단
    };

    // 빈 마스크 생성
    cv::Mat mask = cv::Mat::zeros(image.size(), image.type());

    // 관심 영역(ROI) 마스크 만들기
    std::vector<std::vector<cv::Point>> contours;
    contours.push_back(trapezoid_points);
    cv::fillPoly(mask, contours, cv::Scalar(255, 255, 255));

    // 마스크 적용하여 ROI 추출
    cv::Mat roi_image;
    cv::bitwise_and(image, mask, roi_image);

    // 결과 이미지 표시
    cv::imshow("Original Image", image);
    cv::imshow("ROI Mask", mask);
    cv::imshow("Masked ROI Image", roi_image);

    // 키 입력 대기
    cv::waitKey(0);
    cv::destroyAllWindows();

    return 0;
}
```

- 먼저 이미지를 로드하고 Trapzoid 형태의 ROI를 설정한다. 이때 cv::point 함수를 통해 이미지 비율에 맞게 설정한다.

- 다음 흑색의 빈 마스크를 생성한다. 이후 비트 연산에 활용하자.

- 이때 벡터로 cv::Point를 행렬형태로 만들고 contours에 저장한다. 이전에 생성한 trapezoid를 집어 넣음으로써 다각형 ROI를 만들 수 있다.

- 이전 설정한 ROI를 바탕으로 ROI 마스크를 생성한다. cv::fillPoly()를 사용하여 사다리꼴 형태로 마스크 영역을 채우고, cv::Scalar(255, 255, 255)를 통해 마스크에서 해당 영역을 흰색으로 지정한다.
- 이렇게 ROI 마스크가 생성되면 bitwise_and 함수를 통해 마스크 적용을 할 수 있다.

5. canny, houghline, labeling 함수

```cpp
#include <opencv2/opencv.hpp>
#include <vector>

int main() {
    // 이미지 로드
    cv::Mat image = cv::imread("example.jpg", cv::IMREAD_GRAYSCALE);
    if (image.empty()) {
        printf("이미지를 불러올 수 없습니다.\n");
        return -1;
    }

    // 캐니 에지 검출 (가장자리 검출)
    cv::Mat edges;
    cv::Canny(image, edges, 100, 200);

    // 허프 변환을 이용한 직선 검출
    std::vector<cv::Vec4i> lines;
    cv::HoughLinesP(edges, lines, 1, CV_PI / 180, 50, 50, 10);

    // 라벨링을 위한 바이너리 이미지
    cv::Mat binary;
    cv::threshold(edges, binary, 128, 255, cv::THRESH_BINARY);

    // 라벨링 (Connected Components Analysis)
    cv::Mat labels, stats, centroids;
    int num_objects = cv::connectedComponentsWithStats(binary, labels, stats, cent

    // 결과 이미지에 검출된 라인과 라벨링 정보 표시
    cv::Mat output;
    cv::cvtColor(image, output, cv::COLOR_GRAY2BGR);

    // 허프 변환으로 검출된 직선 그리기
    for (size_t i = 0; i < lines.size(); i++) {
        cv::Vec4i l = lines[i];
        cv::line(output, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]), cv::Scalar(
    }

    // 라벨링된 객체 중심 표시
    for (int i = 1; i < num_objects; i++) {
        int x = centroids.at<double>(i, 0);
        int y = centroids.at<double>(i, 1);
        cv::circle(output, cv::Point(x, y), 5, cv::Scalar(0, 0, 255), -1);
    }

    // 결과 이미지 출력
    cv::imshow("Original Image", image);
    cv::imshow("Canny Edges", edges);
    cv::imshow("Detected Lines and Labels", output);

    cv::waitKey(0);
    cv::destroyAllWindows();

    return 0;
}
```

- **캐니 에지 검출**: cv::Canny() 함수를 사용하여 이미지에서 가장자리를 검출한다. 100과 200은 낮은 임곗값과 높은 임곗값으로, 조정에 따라 감지되는 가장자리의 민감도가 달라진다

- **허프 변환 직선 검출**: cv::HoughLinesP() 함수를 사용하여 캐니 에지 검출된 결과에서 직선을 검출한다.

- lines는 직선 정보를 저장하며, 각 직선은 [x1, y1, x2, y2] 형태로 저장된다.

- **라벨링**: cv::connectedComponentsWithStats() 함수를 사용해 서로 연결된 픽셀을 그룹화하여 라벨링한다.

- labels는 각 픽셀의 라벨 값을 담고, stats는 객체의 위치와 크기를 담고 있으며, centroids는 각 객체의 중심 좌표를 담고 있다.

- 검출된 직선을 **녹색 선**으로 표시하고, 라벨링된 객체의 중심을 **빨간색 점**으로 표시하여 결과를 output 이미지에 나타낸다.

- **결과 이미지 출력**: 원본 이미지, 캐니 에지 검출 결과, 허프 변환으로 검출된 직선과 라벨링 정보를 포함한 이미지를 화면에 표시한다.