

codingOn X posco

K-Digital Training 신재생에너지 활용 IoT 과정

OpenCV

Matplotlib과 OpenCV

- 이미지 처리와 데이터 분석과의 통합

- OpenCV로 이미지를 처리한 뒤 Matplotlib로 결과를 시각화하면 더 직관적으로 처리 결과를 확인할 수 있음

- 색상 표현 방식 차이 극복

- OpenCV는 기본적으로 이미지를 **BGR 형식**으로 다루며, Matplotlib은 **RGB 형식**을 사용. Matplotlib를 통해 OpenCV 이미지를 쉽게 변환하고 디스플레이 가능

- OpenCV의 한계 극복

- OpenCV의 imshow는 간단한 이미지 표시만 가능. Matplotlib로 다중 이미지 비교, 축 조정, 제목 설정 등 고급 시각화 기능 가능

Matplotlib과 OpenCV

- 사용방법

```
import cv2
import matplotlib.pyplot as plt

# 이미지 읽기
img_bgr = cv2.imread('test.png')

# OpenCV는 BGR 사용 -> RGB로 변환
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

# Matplotlib로 이미지 표시
plt.imshow(img_rgb)
plt.axis('off') # x축, y축 눈금 제거
plt.show()
```

Matplotlib과 OpenCV

- Matplotlib에서 이미지를 표시하는 함수
- imshow(데이터, [옵션])
 - 데이터 : 표시할 이미지 데이터 (numpy 배열). 2D 배열(그레이스케일) 또는 3D 배열 (RGB, RGBA) 가능
 - cmap : 컬러 맵. 이미지 데이터를 색상으로 변환하는 데 사용. ('gray', 'viridis', 'hot')
 - norm : 데이터 값을 정규화하는 데 사용(matplotlib.colors.Normalize 객체)
 - interpolation : 이미지를 확대/축소할 때 보간법 지정 ('nearest', 'bilinear', 'bicubic' 등)
 - aspect 이미지의 종횡비 설정. ('auto', 'equal', 또는 특정 숫자)
 - vmin, vmax 데이터의 최소값과 최대값을 지정하여 시각화 범위를 조정.

이미지 전처리

이미지 필터링 및 노이즈 제거

- 노이즈란?

- 이미지에 존재하는 원하지 않는 랜덤한 변형(예: 밝기 변화, 색상 왜곡 등)
- 일반적으로 촬영 과정에서 조명, 센서 한계, 압축 과정 등으로 인해 발생

- 이미지에서 중요한 특징(엣지, 코너 등)을 추출할 때, **노이즈로 인해 잘못된 정보가 포함될 수 있음**

- 노이즈를 제거하면 이미지 데이터가 더 균일해지므로 압축 효율이 좋아지고, 데이터 전송 시 불필요한 정보가 줄어듦

이미지 필터링 및 노이즈 제거

- 블러링 기법 : 평균 블러, 가우시안 블러, 미디언 블러
- 평균 블러링 : `blur(이미지, k사이즈(넓이, 높이))`
 - 이미지에서 특정 영역의 픽셀 값을 주변 픽셀들의 평균값으로 대체하여 노이즈를 줄이는 방법
 - 매끄럽고 부드러운 이미지를 얻는 데 유용 함

```
# 평균 블러링 적용
blurred = cv2.blur(img, (5, 5)) # 큰 값일수록 더 흐려짐

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(blurred, cv2.COLOR_BGR2RGB))
plt.axis('off')
```


이미지 필터링 및 노이즈 제거

- 가우시안 블러링 : `GaussianBlur(이미지, k사이즈(넓이, 높이)(3.홀수), x방향표준편차)`
 - 평균 블러링보다 자연스러운 결과를 제공

```
# 가우시안 블러링 적용
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0) # sigmaX=0: 자동 계산

# 시각화
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.axis('off')
```

이미지 필터링 및 노이즈 제거

- 미디언 블러링 : `medianBlur(이미지, k사이즈(홀수))`
 - 커널 내 픽셀 값 중 중앙값으로 해당 픽셀 값을 설정
 - 소금-후추 노이즈(Salt-and-Pepper Noise)를 제거하는 데 효과적

```
# 미디언 블러링 적용
median_blur = cv2.medianBlur(img, 5) # 커널 크기 5

# 시각화
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.axis('off')
```

샤프닝 필터와 엣지 강조

- 샤프닝 필터와 엣지 강조는 이미지의 세부 정보를 강화하거나 특정 특징을 강조하기 위해 사용
- 샤프닝의 역할 : 샤프닝 필터는 이미지에서 흐려진 세부 정보를 강조하여, 보다 **선명하고 뚜렷한 이미지를 생성**
 - 흐릿한 경계와 텍스처를 뚜렷하게 만듦
 - 원본 이미지에서 잃어버린 디테일을 복원하는 데 유용함
- 엣지 강조란? 엣지 강조는 이미지 내 객체의 윤곽선(경계선)을 뚜렷하게 만드는 과정. 객체의 경계를 강조하여 이미지 분석이 용이해짐
- 엣지 강조는 주요 특징(예: 경계, 형태)을 강조하므로, 후속 작업(예: 객체 탐지, 세그먼테이션)이 더 정확해짐

샤프닝 필터와 엣지 강조

- 샤프닝 필터는 이미지의 경계선(엣지)을 강조하여 선명도를 높임
- 보통 엣지를 강조하기 위해 커널 필터를 사용

```
import numpy as np

# 샤프닝 커널 정의
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]]) # 중심 값을 크게 설정하여 엣지를 강조

# 필터 적용
sharpened = cv2.filter2D(img, -1, kernel) # -1: 입력 이미지와 동일한 깊이로 처리

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(sharpened, cv2.COLOR_BGR2RGB))
plt.axis('off')
```

- filter2D(이미지, 출력이미지깊이, 커널(numpy배열))

그레이 스케일 사용

- 영상 및 이미지 전처리에서는 그레이스케일로 변환하는 것이 일반적
- 컴퓨터 비전 알고리즘이 색상 정보보다는 픽셀 강도(밝기)만을 사용하기 때문
- 컬러 이미지는 일반적으로 3개의 채널(RGB 또는 BGR)을 가지며, 각 채널이 개별적인 픽셀 강도를 가짐. 그레이스케일 이미지는 단일 채널로 구성되어 있어 계산량이 크게 감소
- 그레이스케일은 데이터를 간소화하고, 처리 속도를 높이며, 불필요한 색상 정보를 제거하는 데 유리

엣지 검출 및 경계선 추출

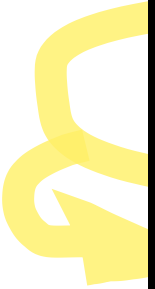
- 엣지 검출은 이미지에서 픽셀 값의 급격한 변화가 발생하는 부분(엣지)을 식별하는 작업
- 물체 윤곽선 추출
 - 이미지를 단순화하여 관심 영역(ROI)의 경계를 강조
 - 물체의 모양이나 크기를 분석하기 위한 전처리 단계로 활용
- 배경 제거 및 특징 추출
 - 배경과 물체의 경계를 분리하여 배경을 제거하거나, 물체의 특징을 강조
- 이미지 분석 및 인식
 - 엣지를 사용하여 물체의 모양, 경계선, 세부 구조를 분석하고 이를 기반으로 물체를 인식

엣지 검출 및 경계선 추출

- Sobel 필터
 - 이미지의 x와 y 방향 기울기를 계산하여 엣지를 검출(1차 미분)
 - 특정 방향(가로, 세로)의 엣지 검출에 유용
- Laplacian 필터
 - 2차 미분을 통해 엣지를 검출
 - 모든 방향의 엣지를 한 번에 검출
 - 물체의 경계를 빠르게 파악
- Canny 엣지 검출
 - 가장 널리 사용되는 엣지 검출 알고리즘
 - 강한 엣지와 약한 엣지 간의 연결을 통해 주요 엣지를 검출
 - 고정밀 객체 탐지 및 추적에 유리하며 특정 물체의 윤곽을 강조하여 물체를 분리

엣지 검출 및 경계선 추출

- 예제코드



```
# Sobel 엣지 검출
sobel_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3) # x 방향 미분
sobel_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3) # y 방향 미분

# Sobel 결과 결합
sobel_combined = cv2.magnitude(sobel_x, sobel_y)

# Laplacian 엣지 검출
laplacian = cv2.Laplacian(img, cv2.CV_64F)

# Canny 엣지 검출
edges = cv2.Canny(img, 100, 200) # 임계값 설정 (100, 200)
```

- magnitude()는 벡터의 크기(벡터의 절대값)를 계산하는 데 사용(x, y)
- CV_64F: 64비트 부동소수점, 더 높은 정밀도 제공

컨투어 탐지 및 활용

- 컨투어는 같은 값의 픽셀로 이루어진 경계선으로, 물체의 윤곽선을 나타냄
- 실무에서 객체 탐지, 추적, 이미지 분석 등 다양한 응용 사례에 필수
- 물체의 모양 및 크기 분석
 - 컨투어를 통해 물체의 면적 (`cv2.contourArea`), 중심 좌표, 둘레 길이 (`cv2.arcLength`), 모양(타원, 사각형 등)을 계산할 수 있음
- 물체 탐지
 - 컨투어를 추출하여 특정 조건(면적, 모양)에 부합하는 물체를 탐지 또는 물체를 추적하거나 인식
- 이미지 분할 및 전처리
 - 컨투어를 사용하여 관심 영역(ROI)을 분리하거나 배경과 물체를 분리함

컨투어 탐지

- findContours(이미지, 컨투어 검색 모드, 컨투어 근사화 방법)
- 컨투어 검색 모드
 - cv2.RETR_EXTERNAL: 외부 윤곽선만 탐지
 - cv2.RETR_TREE: 모든 컨투어를 계층적 구조로 탐지
- 컨투어 근사화 방법:
 - cv2.CHAIN_APPROX_SIMPLE: 컨투어를 압축하여 직선만 저장
 - cv2.CHAIN_APPROX_NONE: 모든 컨투어 포인트를 저장

컨투어 탐지

- 컨투어 윤곽선 그리기
- drawContours(컨투어를 그릴 이미지, 컨투어 정보 리스트, 그릴 컨투어의 인덱스, 컨투어의 색상 (BGR 튜플), 선의 두께)
- 그릴 컨투어의 인덱스
 - -1 : 모든 컨투어를 그림
 - 정수: 특정 인덱스의 컨투어만 그림
- 선의 두께
 - -1 : 컨투어 내부를 채움

이진화

- `threshold()`는 이미지를 이진화하여 픽셀 값을 두 개의 범위로 나눔
- `retval, binary_img = cv2.threshold(src, thresh, maxval, type)`
 - `src`: 입력 이미지 (Grayscale)
 - `thresh`: 임계값(Threshold). 픽셀 값이 이 값을 기준으로 구분
 - `maxval`: 임계값을 넘는 경우 대체될 최대 값 (흰색 값, 보통 255)
 - `type`: 이진화 방식
 - `cv2.THRESH_BINARY`: 픽셀 값이 임계값보다 크면 `maxval`, 작으면 0
 - `cv2.THRESH_BINARY_INV`: `THRESH_BINARY`와 반대. 임계값보다 크면 0, 작으면 `maxval`
 - `cv2.THRESH_TRUNC`: 픽셀 값이 임계값보다 크면 임계값으로 설정, 작으면 그대로 유지
 - `cv2.THRESH_TOZERO`: 픽셀 값이 임계값보다 작으면 0, 크면 그대로 유지
 - `cv2.THRESH_TOZERO_INV`: `THRESH_TOZERO`와 반대. 임계값보다 크면 0, 작으면 그대로 유지

컨투어 탐지

- 예제코드

```
# 이미지 읽기
img = cv2.imread('test_image.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 그레이스케일 변환

# 이진화 처리 회색
X, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# 컨투어 탐지
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# 탐지된 컨투어를 원본 이미지에 그리기
result_img = img.copy()
cv2.drawContours(result_img, contours, -1, (0, 255, 0), 2) # -1: 모든 컨투어 그리기
```

컨투어 특징 계산

- `contourArea` : 컨투어의 영역(픽셀 면적)을 계산
- `moments` : 컨투어의 중심점(중앙 무게중심)을 계산
- `arcLength` : 컨투어의 길이(둘레)를 계산

컨투어 특징 계산

- 예제코드

```
for contour in contours:
    # 1. 면적 계산
    area = cv2.contourArea(contour)
    print(f"Contour Area: {area}")

    # 2. 중심점 계산
    M = cv2.moments(contour)
    if M['m00'] != 0: # 중심점 계산 (m00 = 면적)
        cx = int(M['m10'] / M['m00']) # x 중심
        cy = int(M['m01'] / M['m00']) # y 중심
        print(f"Centroid: ({cx}, {cy})")

        # 중심점 표시
        cv2.circle(result_img, (cx, cy), 5, (0, 0, 255), -1) # 빨간색 점
    else:
        print("Centroid: Undefined (Contour area is 0)")

    # 3. 둘레 계산
    perimeter = cv2.arcLength(contour, True) # True = 폐곡선 여부
    print(f"Arc Length: {perimeter}")

    # 컨투어 그리기
    cv2.drawContours(result_img, [contour], -1, (0, 255, 0), 2) # 초록색 윤곽선
```

영상 연결

비디오 연결

- 비디오 파일 읽기 및 웹캠 스트리밍

- VideoCapture(source)

source

- source : 비디오 경로 또는 0(웹캠)

```
# 비디오 읽기
cap = cv2.VideoCapture(0) # 0: 웹캠

while cap.isOpened():
    ret, frame = cap.read() # 프레임 읽기
    if not ret:
        break

    # 프레임 표시
    cv2.imshow('Video Stream', frame)

    # 'q'를 누르면 종료
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release() # 비디오 캡처 해제
cv2.destroyAllWindows() # 모든 창 닫기
```

비디오 저장

- VideoWriter(filename, fourcc, fps, frameSize)
 - filename : 저장할 파일 이름
 - fourcc : 코덱 (cv2.VideoWriter_fourcc(*'XVID'))
 - fps : 초당 프레임 수
 - frameSize : 프레임 크기 (너비, 높이 튜플)

```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480)) # 20fps, 크기 640x480

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # 비디오 파일에 프레임 저장
    out.write(frame)

cap.release()
out.release()
```

실습. 손 윤곽선을 감지하고 필터를 추가

- 웹캠에서 실시간 영상 읽기
- 손 윤곽선 감지
 - 그레이스케일 변환: `cv2.cvtColor`를 사용
 - 임계값 처리: `cv2.threshold`로 이진화
 - 컨투어 탐지: `cv2.findContours`를 사용하여 손 윤곽선을 감지
 - 컨투어 그리기: `cv2.drawContours`를 활용
- 필터 추가
 - 샤프닝 필터를 적용하여 윤곽선을 강조
- 원본, 윤곽선(컨투어)감지, 샤프닝필터 영상 3개를 띄워서 제출해주세요

복.습.철.저

수고하셨습니다