산업 컴퓨터 비전 실제

기말 프로젝트

2023254006 이선경

CONTENTS

1. 주제

- 이미지 분석을 통한 항공 사진에서의 비행기 식별 및 추적

2. 주제 선정 이유

현업

Prepar3D 엔진을 사용하여
지상통제장치, 조종석 등 개발

이유

중간 프로젝트에서 비행기와 배경의 기본적 분리를 다루었던 것을 기반으로, 이미지 분석 기술을 통해 항공기 식별과 추적의 정밀도를 향상시키기 위함

1. SIFT/SURF 특징 추출

- 비행기 이미지에서 특징점을 감지하고 설명하는 데 사용되는 알고리즘
- 비행기와 다른 배경 요소의 차별화를 위해 강력한 특징점을 추출하고, 이를 통해 비행기의 정확한 식별과 추적을 수행

2. RANSAC 매칭 정제

- 추출된 특징점 간의 매칭 결과에서 오류를 제거하고 더 안정적인 매칭을 확보하기 위해 사용
- 특징점 매칭 과정에서 발견된 잘못된 매칭을 제거하여 비행기의 위치를 더 정확하게 추적

3. GrabCut

- 이미지 내에서 다양한 객체나 색상을 기반으로 그룹을 형성하고, 각 그룹의 중심으로 클러스터를 형성하는 데 사용
- 비행기와 배경과 같은 큰 차이를 나타내는 영역을 분리하여 비행기만을 더 명확하게 추출

4. K-means 클러스터링

- 사용자가 정의한 전경 및 배경의 초기 추정치를 기반으로 이미지에서 객체를 분리
- RANSAC과 특징 매칭을 통해 식별된 비행기의 위치를 기반으로 정확한 객체 분리를 수행

02. 알고리즘

3. 코드

```python
import cv2
import numpy as np
from sklearn.cluster import KMeans

def extract_features_and_match(image, template):
    # 이미지와 템플릿을 그레이스케일로 변환
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if len(template.shape) == 3:
        template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

    # SIFT 특징 추출
    sift = cv2.xfeatures2d_SIFT.create()
    kp1, des1 = sift.detectAndCompute(image, None)
    kp2, des2 = sift.detectAndCompute(template, None)

    # 특징점 매칭
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    # Lowe's ratio test로 좋은 매칭 필터링
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    # RANSAC를 사용하여 매칭 정제
    if len(good_matches) > 4:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        matchesMask = mask.ravel().tolist()
    else:
        matchesMask = None

    return kp1, kp2, good_matches, matchesMask


def apply_kmeans(image, n_clusters=2):
    # 이미지 픽셀 데이터를 2D로 변환
    Z = image.reshape((-1, 3))
    # float 형으로 변환
    Z = np.float32(Z)
    # K-means 클러스터링 수행
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(Z)
    # 각 픽셀에 클러스터 레이블 적용
    center = np.uint8(kmeans.cluster_centers_)
    res = center[kmeans.labels_.flatten()]
    return res.reshape((image.shape))
```

```python
def grabcut_example(image, rect):
    mask = np.zeros(image.shape[:2], np.uint8)
    bgdModel = np.zeros((1, 65), np.float64)
    fgdModel = np.zeros((1, 65), np.float64)

    # GrabCut 알고리즘 실행
    cv2.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

    # 마스크에서 전경 픽셀 추출
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
    result = image * mask2[:, :, np.newaxis]
    return result


# 이미지 로드 및 처리
image_path = 'C:/data/final_airplane.jpg'
template_path = 'C:/data/final_airplane_template.jpg'
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
template = cv2.imread(template_path, cv2.IMREAD_COLOR)

if image is None or template is None:
    print("이미지 또는 템플릿을 로드할 수 없습니다. 경로를 확인하세요.")
else:
    kp1, kp2, good_matches, matchesMask = extract_features_and_match(image, template)

    # 템플릿 매칭을 통해 찾은 위치로 GrabCut 수행
    x, y, w, h = 150, 50, template.shape[1] + 50, template.shape[0] + 50  # 위치 조정
    rect = (x, y, w, h)
    grabcut_result = grabcut_example(image, rect)

    # 전경과 배경 분리
    foreground = np.zeros_like(image)
    background = np.zeros_like(image)

    mask = cv2.cvtColor(grabcut_result, cv2.COLOR_BGR2GRAY)
    foreground[mask != 0] = image[mask != 0]
    background[mask == 0] = image[mask == 0]

    # K-means 클러스터링 적용
    segmented_foreground = apply_kmeans(foreground)
    segmented_background = apply_kmeans(background)


    # 특징 매칭 결과 시각화
    if matchesMask:
        matched_image = cv2.drawMatches(image, kp1, template, kp2, good_matches, None, matchesMask=matchesMask, flags=2)
        cv2.imshow("Matched Features", matched_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("Not enough matches are found - {}/{}".format(len(good_matches), 10))
```

SIFT/SURF 특징 추출

↓

RANSAC 매칭 정제

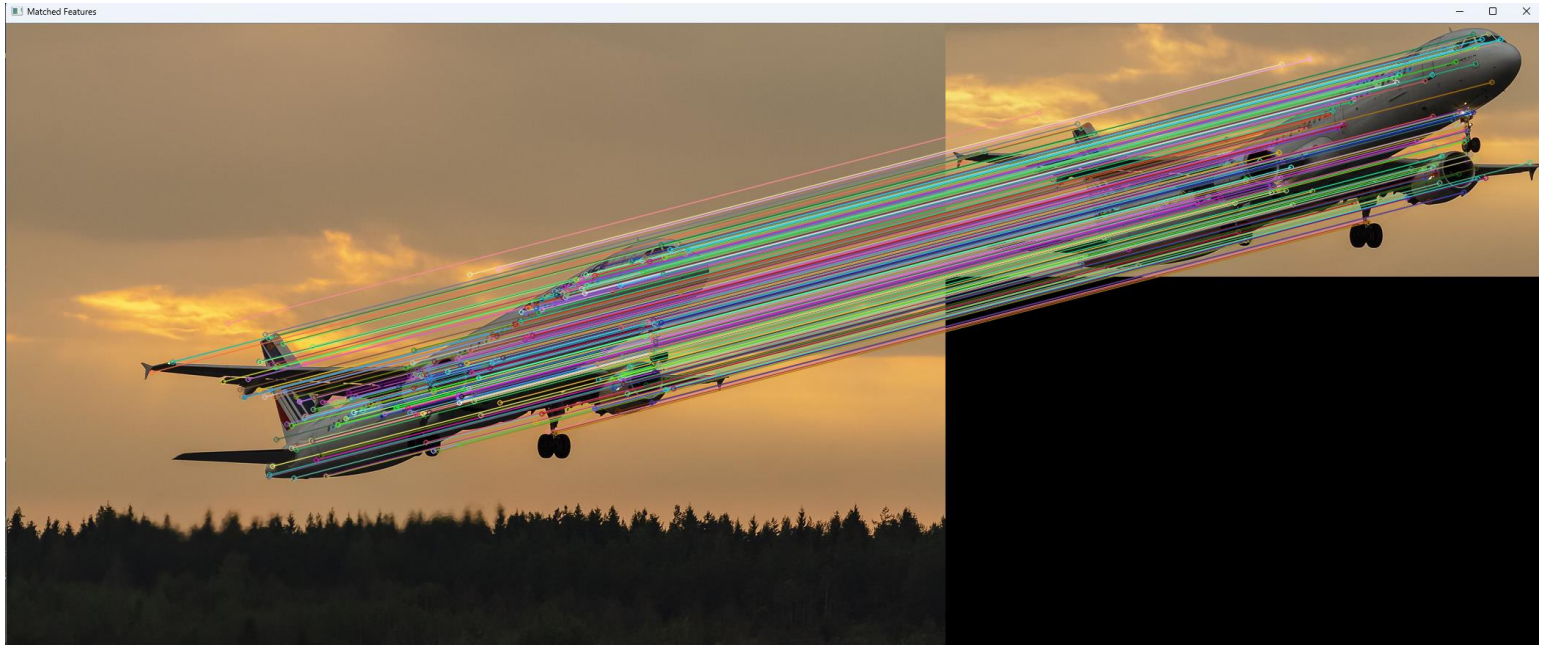
↓

K-means 클러스터링 적용

↓

GrabCut 알고리즘 사용

1. 결과 시각화



- 매칭된 특징점들을 두 이미지 위에 선으로 연결하여 시각화
- 이미지에서 매칭된 점들의 밀집 정도와 위치를 통해 두 이미지 간의 유사성을 판단

2. 고찰

성능 평가

- GrabCut 알고리즘을 통해 전경과 배경을 효과적으로 분리할 수 있었으나,
복잡한 배경이나 비슷한 색상의 전경과 배경 구분에는 한계를 보였음
- SIFT와 RANSAC을 사용한 특징점 매칭은 다양한 조건에서도 견고한 결과를 제공하였으나,
매우 미세한 변화나 높은 노이즈가 있는 환경에서는 정확도가 떨어질 수 있음.
- K-means 클러스터링은 색상 기반 분류에서 효과적이지만,
색상이 유사한 다른 객체들을 오분류 발생

한계점

- 현재 알고리즘은 설정된 파라미터에 매우 의존적이며,
다양한 조건의 이미지에 대해 일반화하기 어려움

향후 개선 방안

- 다양한 환경에서의 실험을 통해 알고리즘의 파라미터를 최적화하여,
더 넓은 조건에서도 효과적으로 작동할 수 있도록 개선

1. 다양한 환경에서의 활용 가능성

- 이미지에서 주요 객체를 강조하거나 배경을 분리하여 시각적으로 인상적인 결과 생성 가능
- 광고, 예술 작업, 게임 개발 등에서 효과적으로 활용될 수 있음

2. 향후 개선 방향

- 정밀한 객체 추출과 배경 처리 기술의 개선을 통해 더욱 정확하고 세밀한 결과를 제공
  이를 위해 추가적인 이미지 처리 기법과 학습 알고리즘의 도입을 고려