

자동차 번호판 검출			
학번	20170766	이름	이건우
제출일	2022. 06. 12	전자 메일	dmz0128@naver.com
주제	자동차 번호판 검출		
실습목표	영상 필터링, 모폴로지 변환, 연결 요소 생성 등 여러 영상처리 과정을 통해 다양한 이미지에 대해 자동차 번호판 영역을 검출한다.		
해결 과정	<ul style="list-style-type: none"> • 해결 과정 <ol style="list-style-type: none"> 1. 컬러 영상을 그레이 스케일 영상으로 변환한다. 2. Morphology(black hat, black hat) 연산을 후보 영역을 강조한다. 3. Gaussian blur을 적용하여 평활화를 수행한다. 4. Threshold를 통해 임계화를 수행한다. 5. 연결 요소를 생성한다. 6. 연결 요소의 넓이, 가로, 세로 길이, 비율을 통해 후보 연결요소를 선별한다. 7. 후보 연결요소 사이의 넓이 차이, 너비, 높이 차이, 각도 차이 통해 인접한 연결 요소들을 카운트하여 3개 이상 인접해 있으면 번호판 글자 영역으로 간주한다. 8. 번호판 글자 영역을 통해 일정 너비와 높이의 간격을 둔 번호판 영역을 만든다. 9. 번호판 영역이 여러 개 나왔을 때 영역을 연결하여 확장해주고, 번호판 영역이 하나도 나오지 않았을 때는 Gaussian blur의 커널 사이즈를 재조정하여 번호판 영역이 검출되거나 커널 사이즈가 1이 될 때까지 번호판 영역 검출을 반복한다. 10. 9번까지의 과정을 지나도 번호판 영역이 검출되지 않는다면, 자동차의 이동 동선 영역 전체를 포함하는 영역을 번호판 영역으로 간주한다. 11. 원본 영상에서 검출한 자동차 번호판 영역을 표시한 후 결과 영상을 저장한다. 12. 지정한 폴더에 포함된 모든 영상에 대해 1~11단계를 반복한다. 		
설계 및 구현	<pre># 모폴로지 연산(탑햇, 블랙햇)을 통해 번호판 영역이 강조 되도록 해준다 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (63, 49)) tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, kernel) blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, kernel) gray = cv2.add(gray, tophat) gray = cv2.subtract(gray, blackhat) - Morphology 연산(top hat, black hat)을 통해 밝기 변화 영역과 어두운 부분의 대비를 높여서 번호판 영역이 강조될 수 있도록 한다 blur = cv2.GaussianBlur(gray, ksize=(ksize, ksize), sigmaX=5) thresh = cv2.adaptiveThreshold(blur, maxValue=255.0, adaptiveMethod=cv2.ADAPTIVE_THRESH_MEAN_C, thresholdType=cv2.THRESH_BINARY_INV, blockSize=21, C=9)</pre> <ul style="list-style-type: none"> - Gaussian blur 적용 후 Threshold을 사용하여 임계화를 수행한다. - Gaussian blur의 Kernel 크기가 (9, 9)이고 sigmaX 값이 5일 때 가장 정확도가 높게 나왔다. - Threshold는 적응형 임계처리의 평균값 방법을 사용하고, 블록크기 21, 차감 값 9를 적용했을 때 가장 정확도가 높게 나왔다. 		

```

    _contours, _ = cv2.findContours(
        thresh,
        mode=cv2.RETR_LIST,
        method=cv2.CHAIN_APPROX_SIMPLE
    )

    contours = []
    for contour in _contours:
        x, y, w, h = cv2.boundingRect(contour)

        contours.append({
            'contour': contour,
            'x': x,
            'y': y,
            'w': w,
            'h': h,
            'cx': x + (w / 2),
            'cy': y + (h / 2)
        })

```

- 연결 요소를 생성한 후 값을 찾기 쉽게 딕셔너리로 만들어 준다.

```

possible_contours = []

cnt = 0
for d in contours:
    area = d['w'] * d['h']
    ratio = d['w'] / d['h']

    if area > MIN_AREA \
        and d['w'] > MIN_WIDTH and d['h'] > MIN_HEIGHT \
        and MIN_RATIO < ratio < MAX_RATIO:
        d['idx'] = cnt
        cnt += 1
    possible_contours.append(d)

```

- 생성한 연결요소 중에서 일정 넓이, 너비, 높이, 비율을 만족하는 연결요소를 선택하여 분류한다.
- 최소 넓이: 100, 최소 너비: 6, 최소 높이: 30, 최소 비율: 0.25, 최대 비율: 1.0 일 때 가장 정확도가 높게 나왔다.

```

def find_chars(contour_list):
    matched_result_idx = []

    for d1 in contour_list:
        matched_contours_idx = []
        for d2 in contour_list:
            if d1['idx'] == d2['idx']:
                continue

            dx = abs(d1['cx'] - d2['cx'])
            dy = abs(d1['cy'] - d2['cy'])

            diagonal_length1 = np.sqrt(d1['w'] ** 2 + d1['h'] ** 2)

            distance = np.linalg.norm(np.array([d1['cx'], d1['cy']]))

            if dx == 0:
                angle_diff = 90
            else:
                angle_diff = np.degrees(np.arctan(dy / dx))
            area_diff = abs(d1['w'] * d1['h'] - d2['w'] * d2['h']) / (d1['w'] * d1['h'])
            width_diff = abs(d1['w'] - d2['w']) / d1['w']
            height_diff = abs(d1['h'] - d2['h']) / d1['h']

            if abs(angle_diff) < 10 and abs(area_diff) < 10 and abs(width_diff) < 10 and abs(height_diff) < 10:
                matched_contours_idx.append(d2['idx'])

        matched_result_idx.append(matched_contours_idx)

```

```

        if distance < diagonal_length1 * MAX_DIAG_MULTIPLIER \
            and angle_diff < MAX_ANGLE_DIFF and area_diff < MAX_AREA_DIFF \
            and width_diff < MAX_WIDTH_DIFF and height_diff < MAX_HEIGHT_DIFF:
            matched_contours_idx.append(d2['idx'])

    matched_contours_idx.append(d1['idx'])

    if len(matched_contours_idx) < MIN_N_MATCHED:
        continue

    matched_result_idx.append(matched_contours_idx)

    unmatched_contour_idx = []
    for d4 in contour_list:
        if d4['idx'] not in matched_contours_idx:
            unmatched_contour_idx.append(d4['idx'])

    try:
        unmatched_contour = np.take(contour_list, unmatched_contour_idx)
        recursive_contour_list = find_chars(unmatched_contour)

        for idx in recursive_contour_list:
            matched_result_idx.append(idx)
        break
    except IndexError:
        break

    return matched_result_idx
result_idx = find_chars(possible_contours)

matched_result = []
for idx_list in result_idx:
    matched_result.append(np.take(possible_contours, idx_list))

```

- 각 연결요소와 주위 연결요소를 비교하여 거리가 대각선 길이의 일정한 배수 이내인지, 각도, 넓이, 너비, 높이 차이가 일정 배수 이내인지를 확인하여 조건을 만족한 연결요소를 찾고, 일정 개수 이상 연속으로 조건을 만족한 연결요소의 인덱스를 반환한다.
- 반환된 인덱스에 해당하는 연결요소만을 선택하여 분류한다.
- 대각선 길이의 최대 배수: 5, 각도 차이 최대 배수: 7, 넓이 차이 최대 배수: 0.2, 너비 차이 최대 배수: 0.8, 높이 차이 최대 배수: 0.2, 연속으로 조건을 만족한 연결요소의 최소 개수: 3 일 때 가장 높은 정확도가 나왔다.

```

plate_infos = []
for i, matched_chars in enumerate(matched_result):
    sorted_chars = sorted(matched_chars, key=lambda x: x['cx'])

    plate_cx = (sorted_chars[0]['cx'] + sorted_chars[-1]['cx']) / 2
    plate_cy = (sorted_chars[0]['cy'] + sorted_chars[-1]['cy']) / 2

    plate_width = (sorted_chars[-1]['x'] + sorted_chars[-1]['w'] - sorted_chars[0]['x'])

    sum_height = 0
    for d in sorted_chars:
        sum_height += d['h']

    plate_height = int(sum_height / len(sorted_chars) * PLATE_HEIGHT_PADDING)

    plate_infos.append({
        'x': int(plate_cx - plate_width / 2),
        'y': int(plate_cy - plate_height / 2),
        'w': int(plate_width),
        'h': int(plate_height)
    })

```

- 남은 연결요소들을 번호판의 번호 영역으로 간주하고 너비와 높이에 일정한 간격을 추가하여 사각형을 형태로 합치고 시작 좌표와 너비, 높이를 딕셔너리로 만든다.

	<ul style="list-style-type: none"> - 너비 간격 배수: 1.3, 높이 간격 배수: 2.0 일 때 가장 정확도가 높게 나왔다. <pre> if plate_infos: info = plate_infos[-1] for i in plate_infos: x, y, w, h = i.values() if x < info['x']: info['x'] = x if y < info['y']: info['y'] = y if w > info['w']: info['w'] = w if h > info['h']: info['h'] = h if info['x'] < 0: info['x'] = 0 if info['y'] < 0: info['y'] = 0 else: if ksize > 2: return detect_car_license_plate(img, fname, ksize - 2) else: info = {'x': 120, 'y': 230, 'w': 830, 'h': 470} points = np.concatenate([[info['x']], [info['y']], [info['x']+info['w']], [info['y']+info['h']]]) return points </pre> <ul style="list-style-type: none"> - 번호판 후보 영역들 중 가장 큰 영역을 선택하는 것 보다. 후보 영역들을 포함하는 큰 영역을 만들었을 때 정확도가 높았다. - 음수가 나오는 것을 방지하여, 음수일 경우 0을 대입했다. - 모든 과정을 거쳐도 번호판 후보 영역을 찾을 수 없을 때, Gaussian blur의 커널 사이즈를 2 줄여 다시 검출을 시도하면 조금 더 정확도를 높일 수 있었다. - 마지막으로 도무지 찾지 못했을 때는 자동차의 이동 영역을 가정하여, 이동 영역 전체를 아우르는 큰 사각형을 만들었다. - 만들어진 사각형의 왼쪽 상단 좌표와 오른쪽 하단 좌표를 반환해준다.
실험 결과	<ul style="list-style-type: none"> • 평균 정확도 <pre> (base) leegw ~/Desktop/Work/OpenCV/car_license_plate_detect ↵ main ±+ python accuracy.py -r ref.dat -d detect.dat total number of list: 425 total number of list: 425 Accuracy data in accuracy.dat ===== average Precision: 0.8 average Recall: 0.79 average F1_Score: 0.77 average IOU_Score: 0.66 </pre> <ul style="list-style-type: none"> - F1 스코어 기준 평균 77%의 정확도가 나왔다. <ul style="list-style-type: none"> • 소요 시간

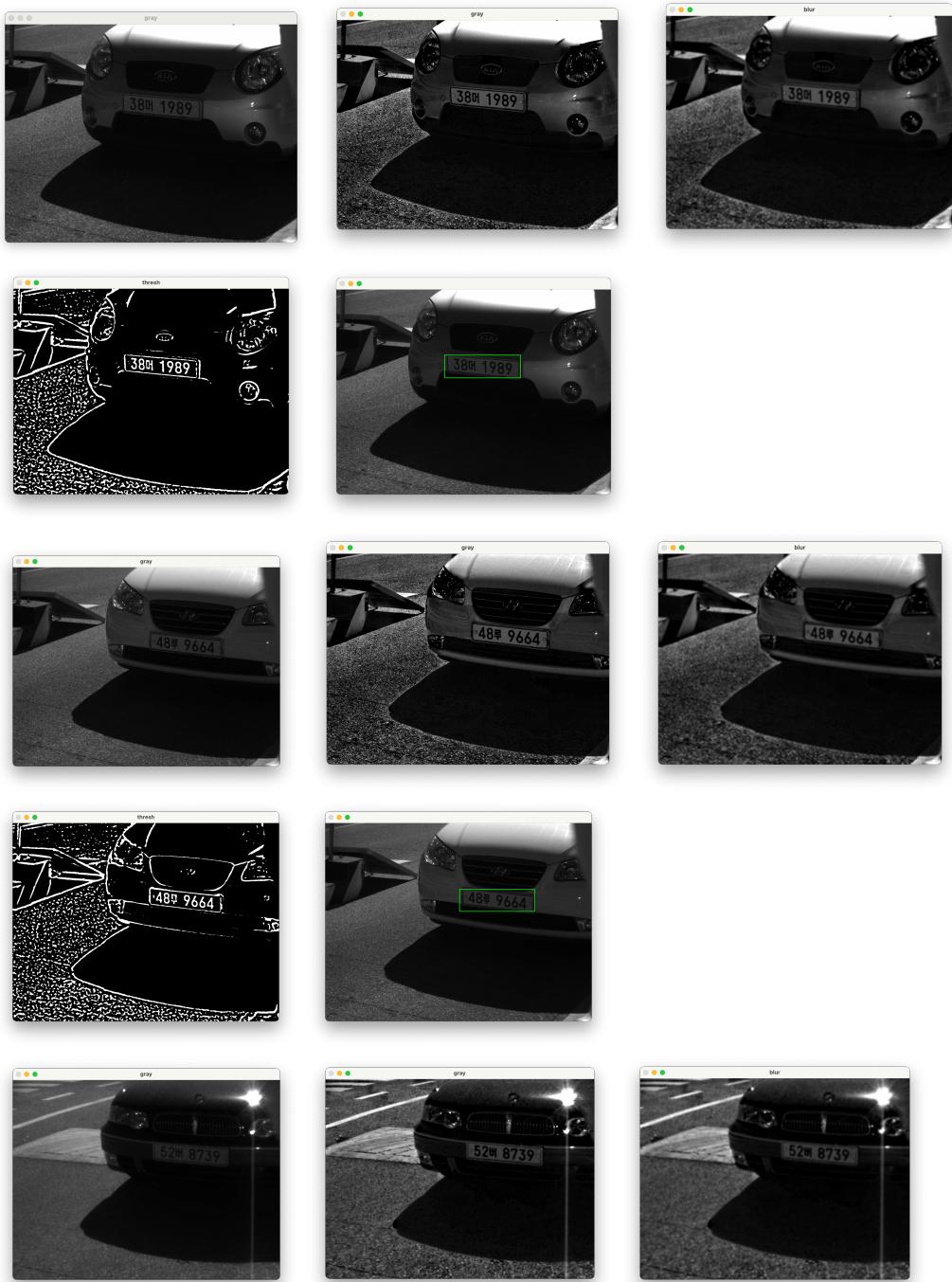
```

dataset/2015031010375623우 5783.jpg 처리 중 ...
dataset/2015031010253135비 3955.jpg 처리 중 ...
dataset/2015031010255685마 8348.jpg 처리 중 ...
dataset/2015031010314462두 9044.jpg 처리 중 ...
dataset/20150310103936경 북 15바 2325.jpg 처리 중 ...
dataset/2015031010463221라 8547.jpg 처리 중 ...
dataset/2015031010585713허 5402.jpg 처리 중 ...
dataset/2015031010541820어 8145.jpg 처리 중 ...
time : 19.742741107940674
(base) leegw ~/Desktop/Work/OpenCV/car_license_plate_detect ↵ main ±+

```

- 모든 영상을 처리하는데 약 19.7초가 소요되었다.

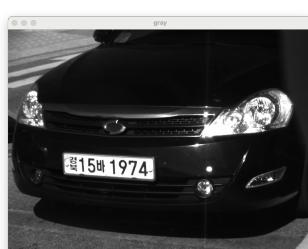
- 정확도 90%이상



정확도 분석

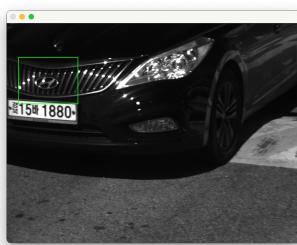
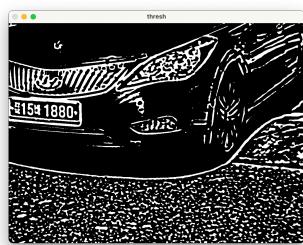
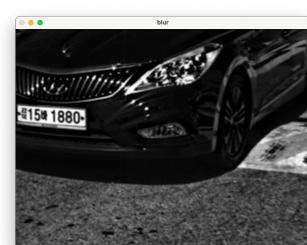
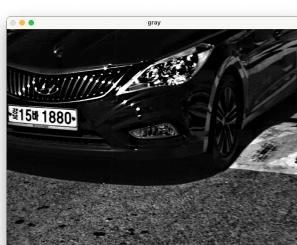
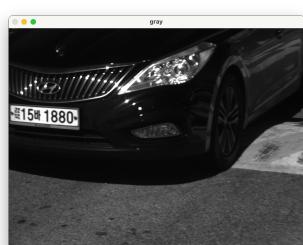
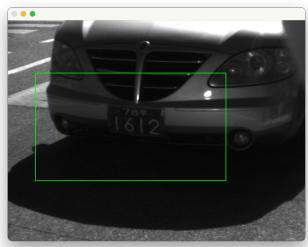


- 정확도 40% 이상 60% 미만





- 정확도 10% 미만



	 <ul style="list-style-type: none"> • 원인 분석
느낀 점	<ul style="list-style-type: none"> - 정확도 평균 90% 이상의 경우, 차량과 번호판의 대비가 뚜렷하고 차량에 빛이 반사되는 부분이 거의 없어 정확하게 번호판 영역이 검출되었다. - 정확도 평균 60% 미만 40% 이상의 경우, 차량 범퍼에 빛이 반사되는 부분이 있어 빛이 반사되는 부분까지 번호판 영역이 확장되어 검출되었다. - 정확도 평균 10% 미만의 경우, 번호판과 차량의 대비가 뚜렷하지 않거나 빛이 심하게 반사되는 부분이 있어 번호판 영역을 제대로 검출하지 못하고 있다. • 결론 - 차량과 번호판의 대비가 클수록, 차량이나 주변에 빛의 반사가 적을수록 번호판 영역이 정확하게 검출되었다.