연구 주제명: 눈동자 인식을 이용한 화상 마우스/키보드

연구 목표: 시선의 위치와 모니터 상 마우스 위치의 mapping 및 눈동자 인식률 향상

연구 요약:

1. 눈동자 인식률이 낮았던 원인 발견 및 개선
   - 눈동자 감지 영역에 눈동자가 완전히 들어오지 않는 경우가 많았음
   -> 개선된 감지 영역 및 이미지 전처리 과정 개선으로 인식률 향상

2. 시선 인식을 기반한 마우스 포인터 좌표 이동
   - 시선의 이동 범위와 마우스 포인터의 이동 범위를 적절히 매칭

5 월 17 일

국민대학교 전자공학부 전자종합설계 I

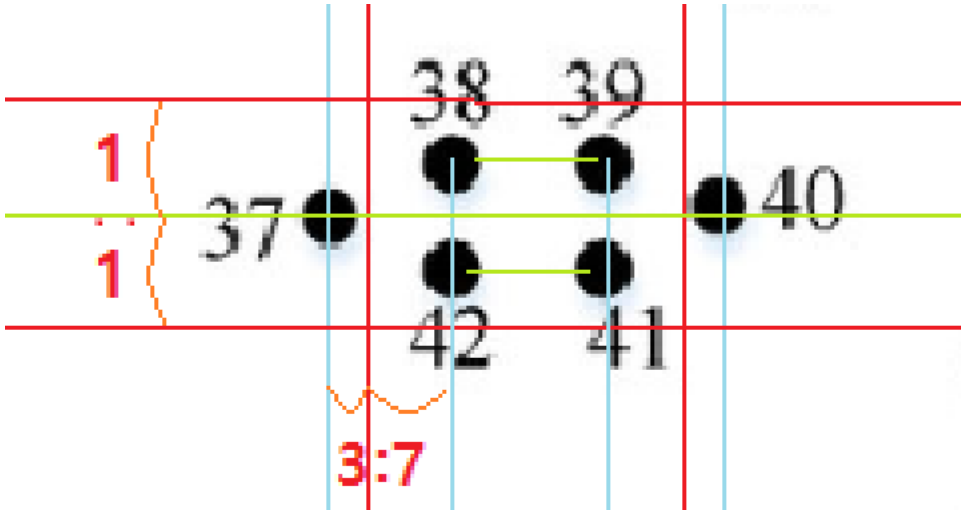
연구원: 김정민(20162934) , 신미나(20201463)

연구 내용

문제점 발견 및 개선

9 주차 프로젝트에서 눈동자 검출을 위해 dlib 의 랜드마크 중 양 눈에 해당하는 6 쌍의 점을 Hexagon mask 을 씌워 영역을 분할하였는데, 이 마스크 내에 눈동자가 완전히 포함되지 않는 문제점을 발견했다. 실시간으로 얼굴의 68 개 랜드마크를 그린 결과를 확인해보니 눈의 랜드마크 범위가 실제 눈의 크기보다 작은 경우가 빈번했다. 따라서 dlib 를 이용한 눈동자 인식률이 낮았던 이유는 낮은 레벨의 영상처리기법(Hough 변환)을 사용했다는 점 뿐만 아니라, "애초에 처리되는 영역에 눈동자가 완전히 들어오지 못했던 점"도 해당됐을 것이다.

이를 해결하기 위해, 눈동자 인식 영역을 새로 구현하였다. 6 개의 눈 랜드마크 좌표를 그대로 사용하지 않고 좌표 사이의 비율을 고려하여 새로운 인식 영역을 사각형으로 구현하였다.



눈동자 인식 영역 계산 비율

위 그림의 빨간선이 눈동자 인식 영역이다.
눈동자의 움직임과 랜드마크의 위치를 실시간으로 확인해 본 결과, 눈동자는 양 옆의 끄트머리로는 잘 이동하지 않고 수직방향으로 랜드마크를 자주 벗어나는 움직임을 보였다. 따라서 기존 인식 영역의 모양을 복잡한 육각형에서 사각형으로 변경하고, 수평방향으로는 좁히고 수직방향으로는 넓혔다.

그리고 눈동자 검출의 정확도를 높이기 위해 이미지 전처리 과정에 히스토그램 평활화 과정을 추가했고, 영역 내 눈동자를 제외한 다른 circle 이 반환되지 못하게 하기 위하여 circle 객체는 오직 한 개 그 이외에 None 값을 저장하여 반환하도록 변경했다.

프로그램 수정사항

- 사용할 라이브러리 추가

```
import cv2
import numpy as np
import dlib
import pyautogui
pyautogui.FAILSAFE = False

# 출력창 크기 설정
output_width, output_height = 1920, 1080

# 웹캠 지원 최대 해상도 = 640 * 480
```

- 눈동자 인식 영역 개선

```
left_eye_pts = []
right_eye_pts = []

for eye in eyes :
    mid_side_h = np.mean([landmarks.part(eye[0]).y, landmarks.part(eye[3]).y])
    up_side_dd = 2 * abs(mid_side_h - np.mean([landmarks.part(eye[1]).y, landmarks.part(eye[2]).y]))
    up_side = mid_side_h - up_side_dd
    low_side_dd = 2 * abs(mid_side_h - np.mean([landmarks.part(eye[4]).y, landmarks.part(eye[5]).y]))
    low_side = mid_side_h + low_side_dd
    mid_side_w1 = np.mean([landmarks.part(eye[1]).x, landmarks.part(eye[5]).x])
    mid_side_w2 = np.mean([landmarks.part(eye[2]).x, landmarks.part(eye[4]).x])
    left_side_dd = 0.3 * abs(mid_side_w1 - landmarks.part(eye[0]).x)
    right_side_dd = 0.3 * abs(mid_side_w2 - landmarks.part(eye[3]).x)
    left_side = landmarks.part(eye[0]).x #+ left_side_dd
    right_side = landmarks.part(eye[3]).x #- right_side_dd
    if len(left_eye_pts) == 0 :
        left_eye_pts.append((left_side, up_side)) # 좌상단
        left_eye_pts.append((right_side, up_side)) # 우상단
        left_eye_pts.append((right_side, low_side)) # 우하단
        left_eye_pts.append((left_side, low_side)) # 좌하단
    else :
        right_eye_pts.append((left_side, up_side)) # 좌상단
        right_eye_pts.append((right_side, up_side)) # 우상단
        right_eye_pts.append((right_side, low_side)) # 우하단
        right_eye_pts.append((left_side, low_side)) # 좌하단
```

위 그림과 동일하게 계산하여 구현하였다.

Left_eye_pts 와 right_eye_pts 는 각 눈의 눈동자를 검출할 사각형 경계영역으로, HoughCircles 알고리즘은 이 영역 내에서만 눈동자를 검출한다.

- 시선 좌표와 마우스의 좌표를 매칭

```python
# 마우스 포인터가 이동할 좌표를 반환
def get_point_to_move(rect_coords, point_coord):
    # 사각형 내부의 포인트 좌표 추출
    point_x, point_y = point_coord[0] , point_coord[1]

    # 사각형의 좌측 상단과 우측 하단 꼭지점 좌표
    rect_x = rect_coords[0].x
    rect_y = rect_coords[0].y
    rect_width = rect_coords[1].x - rect_x
    rect_height = rect_coords[1].y - rect_y

    # 주어진 사각형 영역에서의 상대 좌표
    relative_x = point_x - rect_x
    relative_y = point_y - rect_y

    # 모니터 화면에서의 대응 좌표 계산
    target_x = int(output_width * (relative_x / rect_width))
    target_y = int(output_height * (relative_y / rect_height))

    # 좌표 반환
    return [target_x, target_y]
```

시선의 움직임을 모니터 상 마우스의 움직임에 매칭하여 마우스의 목적지 좌표를 반환하는 함수이다.
파라미터인 'rect_coords'는 시선 좌표의 바운더리로, 간단히 dlib 가 주는 눈 랜드마크 두 개로 구성된다. (시선 좌표는 '눈동자(원)의 중심점'이므로, 위에서 구현한 눈동자가 기준의 사각 영역을 그래도 사용할 경우 시선 좌표를 정확히 다루기에 무리가 있다.) 'point_coord'는 시선의 좌표이고 'output_width', 'output_height'는 데스크탑 모니터의 사이즈이다.
핵심은 (시선 바운더리) : (시선 좌표) = (모니터 바운더리) : (마우스 목적좌표) 비례식에 맞춰 적절히 구현하였다.

결과

지난 번보다 눈동자 인식률이 눈에 띄게 향상되었다. 그리고 Pyautogui 를 이용한 시선 좌표에 따라 마우스 포인터를 이동시키는 것 또한 실행해 볼 수 있었으나, 매 프레임마다 영상의 노이즈 또한 매번 랜덤하게 섞이기 때문에 눈동자는 가만히 있는데 시선 좌표가 이리저리 튀는 현상이 나타났다. 또한 코드상으론 마우스 포인터가 이동하는 영역에 제한이 없지만 실제로는 마우스가 화면의 가운데 부분에서만 이동하는 모습을 보였다.

개선할 점 및 다음 목표

사람이 눈동자를 움직이지 않고 가만히 정지할 때 시선 좌표 또한 움직이지 않도록 수정하는 것이 좋겠다. 그리고 시선이 정면을 응시할 때에는 문제가 되지 않지만 시야의 끄트머리를 바라볼 때 눈동자가 원 모양이 아니게 되어 시선 인식을 하지 못하고, 따라서 마우스 포인터가 모니터의 경계면으로 이동하지 못하는 문제점을 개선해야 한다. 또한, 저번과 마찬가지로 눈동자 인식률을 높이기 위해 어떤 방법을 사용할 것인지 고민해봐야 할 것이다.