2조:: 내가 2정도야 팀

FACEBOOKS 프로젝트

2024.09.04

박준수 최재원 권시우 한태섭

2조

Contents

프로젝트 개요

동기

카메라 화면에 가상 요소를 더해 사용자에게 창의적이고 유쾌한 사진 촬영 경험을 제공

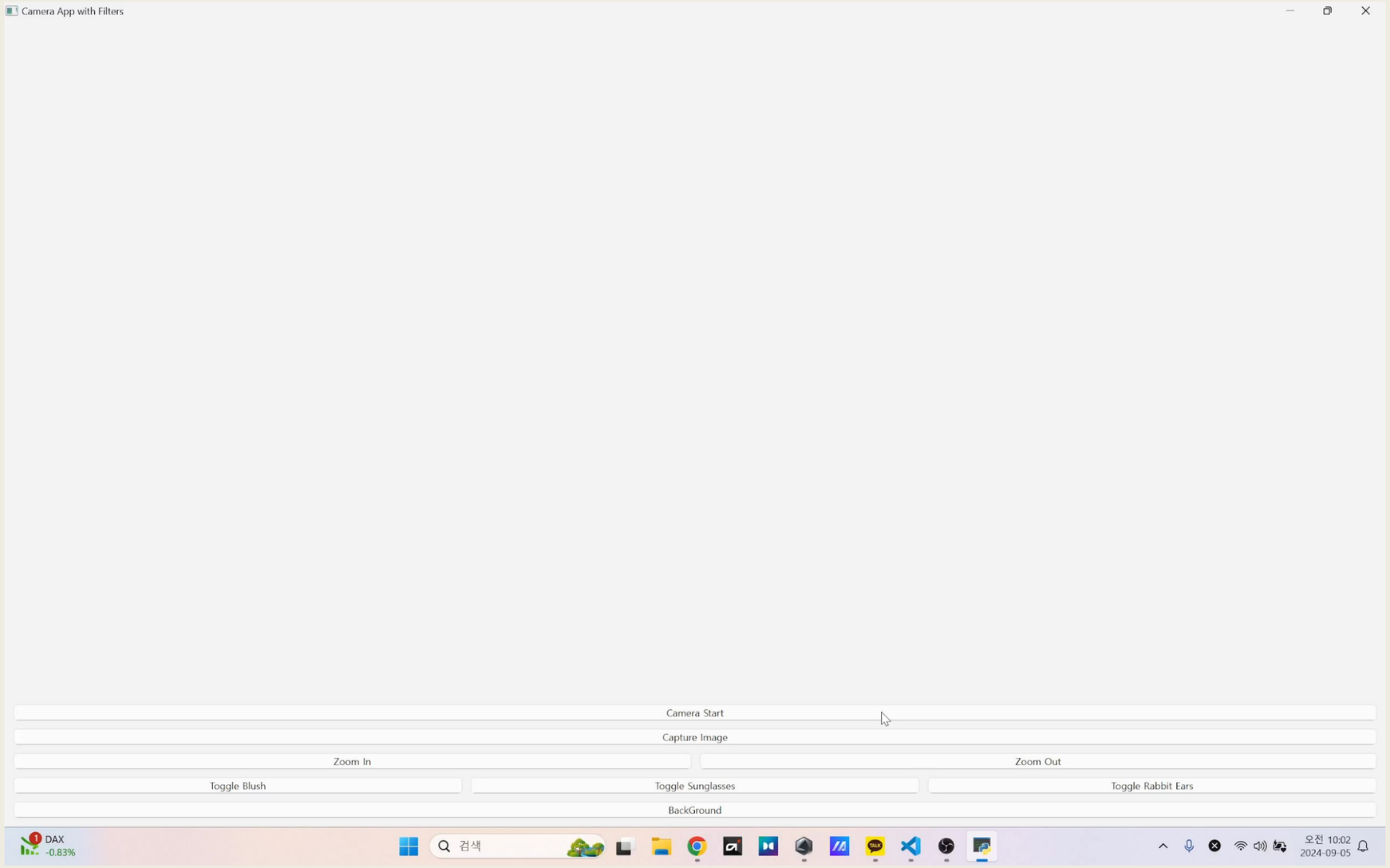
목표

영상처리 기법 사용하여 기존의 카메라 어플의 기능 및 부가적인 기능 (사진 전송 서비스) 구현

프로젝트 팀 정보

프로젝트명	FACEBOOKS	
팀명	내가 ‘2’ 정도야	
이름	담당	역 할
박준수	팀장	토끼 귀 필터 구현, 리팩토링
최재원	팀원	선글라스 필터, 캡쳐 기능 구현
권시우	팀원	Blush, 배경 변경 효과 구현
한태섭	팀원	GUI 및 이미지 출력 기능 구현

시연 영상


Camera App with Filters
Camera Start
Capture Image
Zoom In
Zoom Out
Toggle Blush
Toggle Sunglasses
Toggle Rabbit Ears
BackGround
DAX
-0.83%
검색
오전 10:02
2024-09-05

설계 사양서 - 주 사용 라이브러리 : PySide6, Opencv2, MediaPipe

~~Solutions~~

PySide6

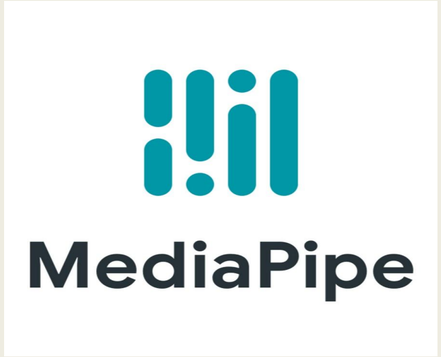
- **GUI 애플리케이션 제작을 위한 라이브러리.**
  Python 바인딩을 한 Qt 프레임워크.
- **크로스 플랫폼** 지원 (Windows, macOS, Linux).
- 창, 버튼, 레이아웃 등 **다양한 위젯**을 제공.
- **이벤트 신호** 및 슬롯 메커니즘으로 사용자 상호작용 처리.
- **멀티쓰레딩 지원, 애니메이션 효과 지원**
- Qt Designer 지원; GUI 디자인 도구

Opencv2

- **컴퓨터 비전 라이브러리**. 이미지 및 영상 처리 기능
- **C++, Python, Java** 등 다양한 언어에서 사용 가능
- 객체 탐지, 얼굴 인식, 모션 추적 기능 제공
- **이미지 변환**, **필터링**, **특징 (**모서리, 윤곽 등) **추출** 같은 이미지 처리 기능 포함
- **GPU 가속**을 지원해 고성능 실시간 처리에 최적화됨.
  머신러닝 모델과의 통합에 유리.

설계 사양서 - 주 사용 라이브러리 : PySide6, Opencv2, MediaPipe Solutions

Google AI Edge 에서 만든 **오픈소스 프로젝트**



Type 1: **MediaPipe Framework:**

- 효율적인 On-Device 머신러닝 파이프라인 구축에 사용되는 저수준 구성 요소

Type 2: **MediaPipe Solutions:**

- 미리 만들어진 고수준 솔루션으로 복잡한 구현 없이 쉽게 컴퓨터 비전 작업을 수행 가능
  얼굴 탐지, 얼굴 랜드마크 탐지, **포즈 추적, 객체 탐지** 등 다양한 이미지 및 비디오 분석 솔루션 제공
- **고성능의 실시간 처리 기능**과 **크로스 플랫폼 지원** (모바일, 데스크탑, 웹에서 실행 가능)

설계 사양서 - Face Landmark Detection

1. **MediaPipe BlazeFace (Short Range)**
   Face Detection

   - **Short-range model** that works **best for faces within 2 meters** from the camera
   - **Full-range model** that works **best for faces within 5 meters** from the camera

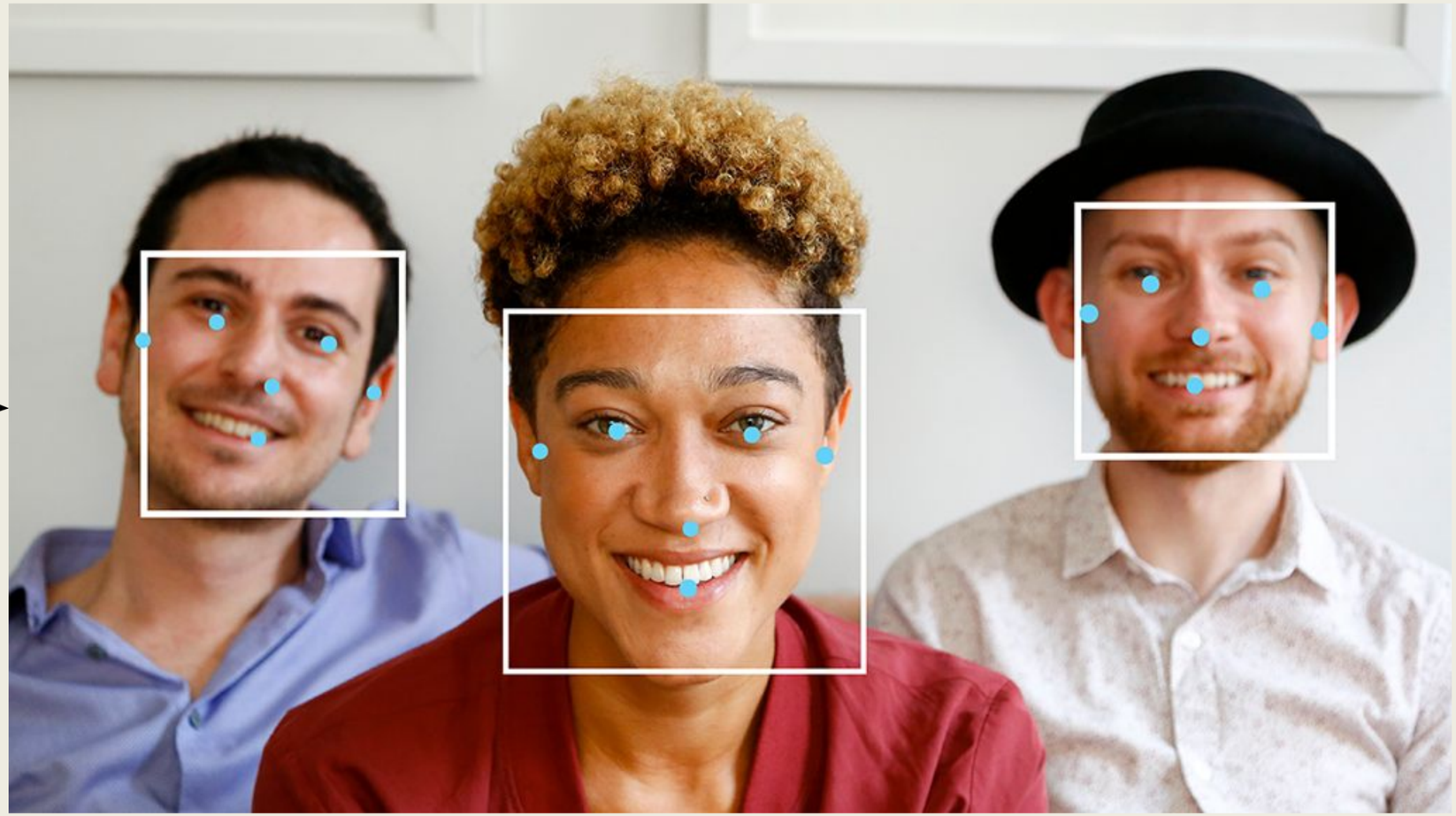
2. **MediaPipe Face Mesh V2**
   Face Landmarks Detection

설계 사양서 - Face Landmark Detection

- **MediaPipe BlazeFace (Short Range)**
  Face Detection

Mapping of 6 facial keypoint coordinates

6 points

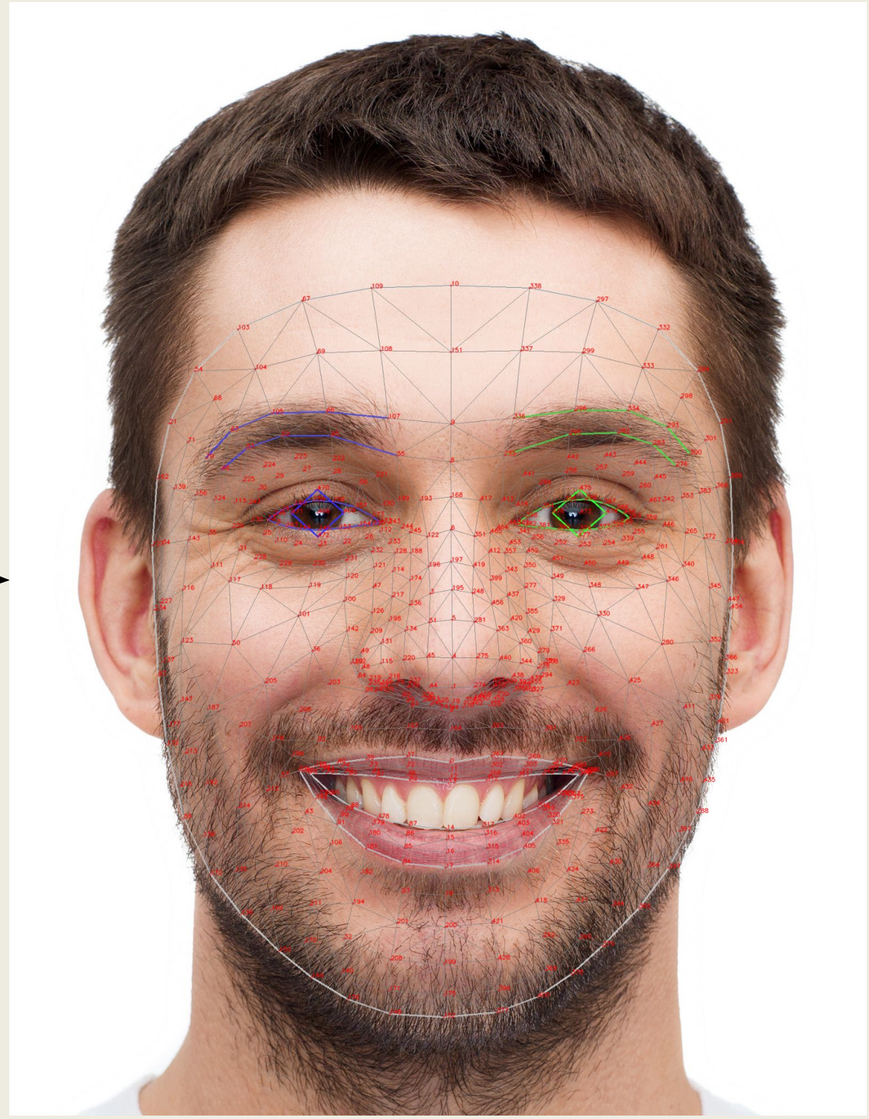


설계 사양서 - Face Landmark Detection

- **MediaPipe Face Mesh V2**
  Face Landmarks Detection

478 points

Mapping of facial landmarks



설계 사양서 - Face Landmark Detection

- **MediaPipe BlazeFace (Short Range)** Face Detection	- **MediaPipe Face Mesh V2** Face Landmarks Detection
● **Return:** For each detected face, returns: ● Facial bounding box coordinates ● **6 approximate facial keypoint coordinates:** ○ Left eye (from the observer's point of view) ○ Right eye ○ Nose tip ○ Mouth ○ Left eye tragion (이주; 귓불 바로 위에 있는 작은 돌기) ○ Right eye tragion ● Detection confidence score	● **Return:** ● **Facial surface** represented as **478 3D landmarks** flattened into a 1D tensor: (x1, y1, z1), (x2, y2, z2), … … as pixel coordinates ● **Face flag** indicating the likelihood of the face being present in the input image. ● [option] **blendshapes** …

설계 사양서 - Face Landmark Detection

- **MediaPipe BlazeFace (Short Range)**
  Face Detection

- **Intended Uses:**
  - Detecting **prominently displayed faces** within images or **videos captured** by a **smartphone camera.**

- **Limitations:**
  - The model struggles with detecting faces that are **not front-facing** or are **too far from the camera**, **especially more than 2 meters away**.

  - The model is biased towards **detecting larger, prominent faces** and may not perform well on faces with **different orientations** or **smaller sizes**.

  - The model's performance can degrade in **poor lighting,** with **motion blur**, or when **faces are overlapping.**

설계 사양서 - Face Landmark Detection

- **MediaPipe Face Mesh V2**
  Face Landmarks Detection

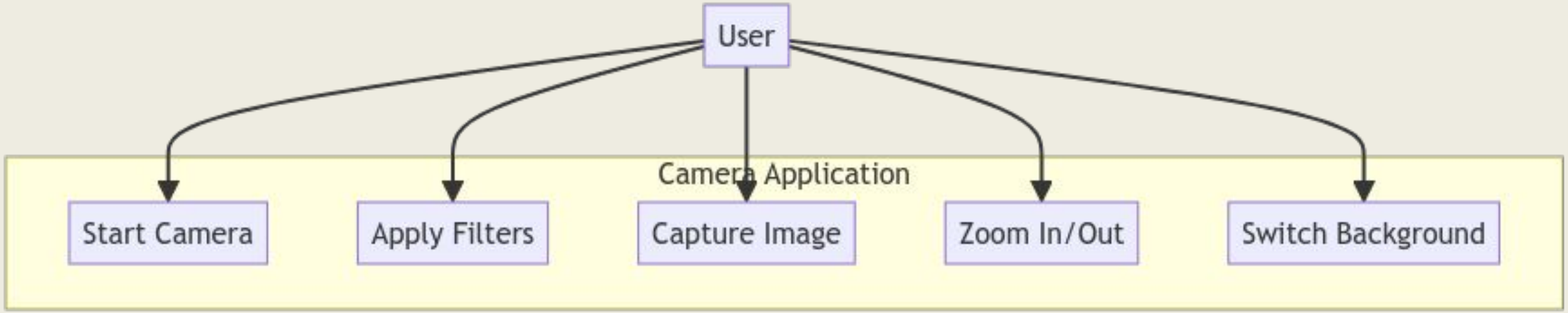
- **Intended Uses:**
  - Designed for use with **front-facing cameras** and is particularly well-suited for AR (augmented reality) applications

- **Limitations:**
  - The model is designed for **close-up, front-facing selfies** and cannot accurately detect faces that are at **extreme angles** or too **far from the camera**.

  - The model's accuracy: performs more stably in well-lit environments, **extreme lighting** or **facial occlusion** can still negatively impact its performance.

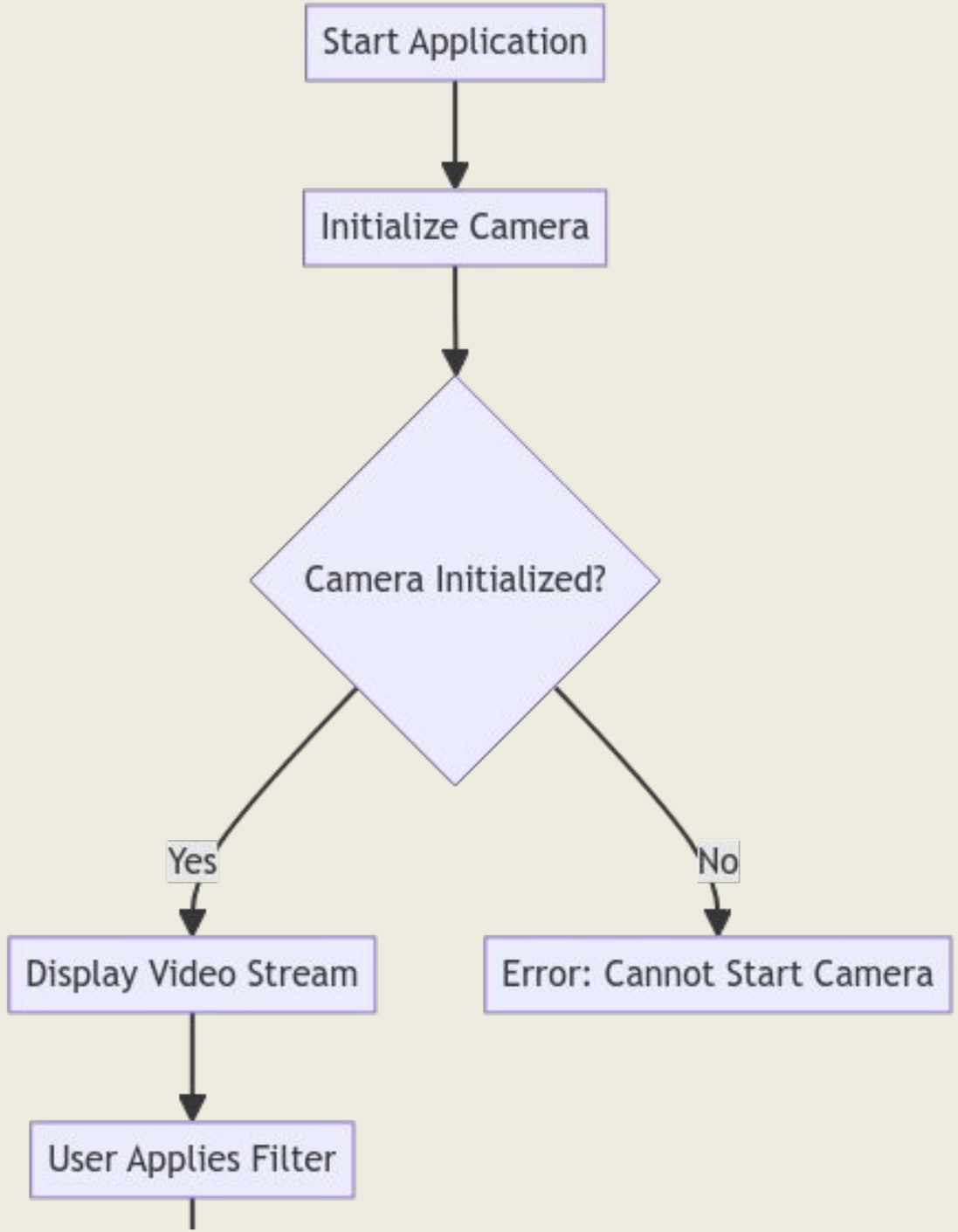
설계 사양서 - Usecase Diagram

-

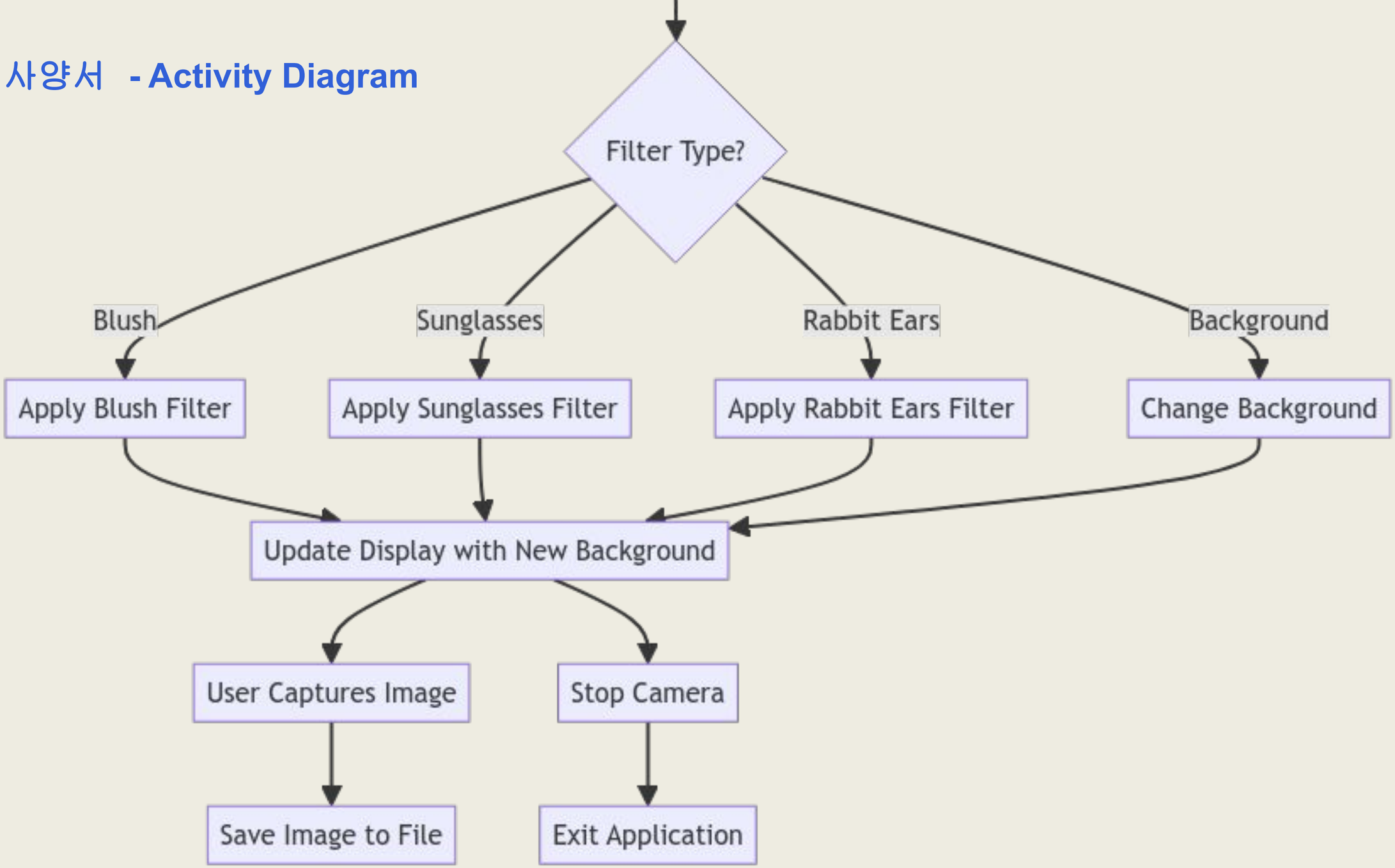


설계 사양서 - Activity Diagram

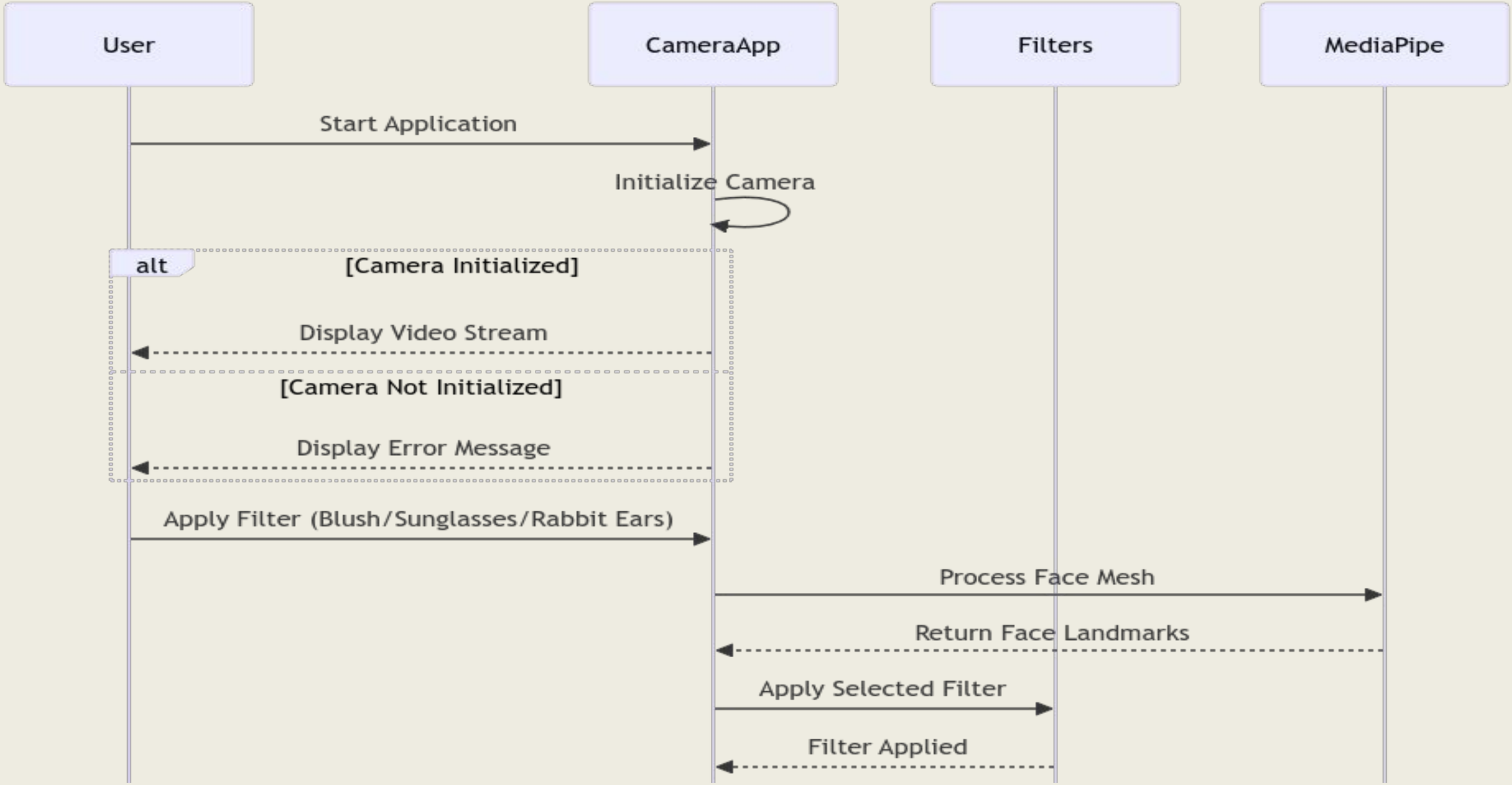
-



설계 사양서 - Activity Diagram

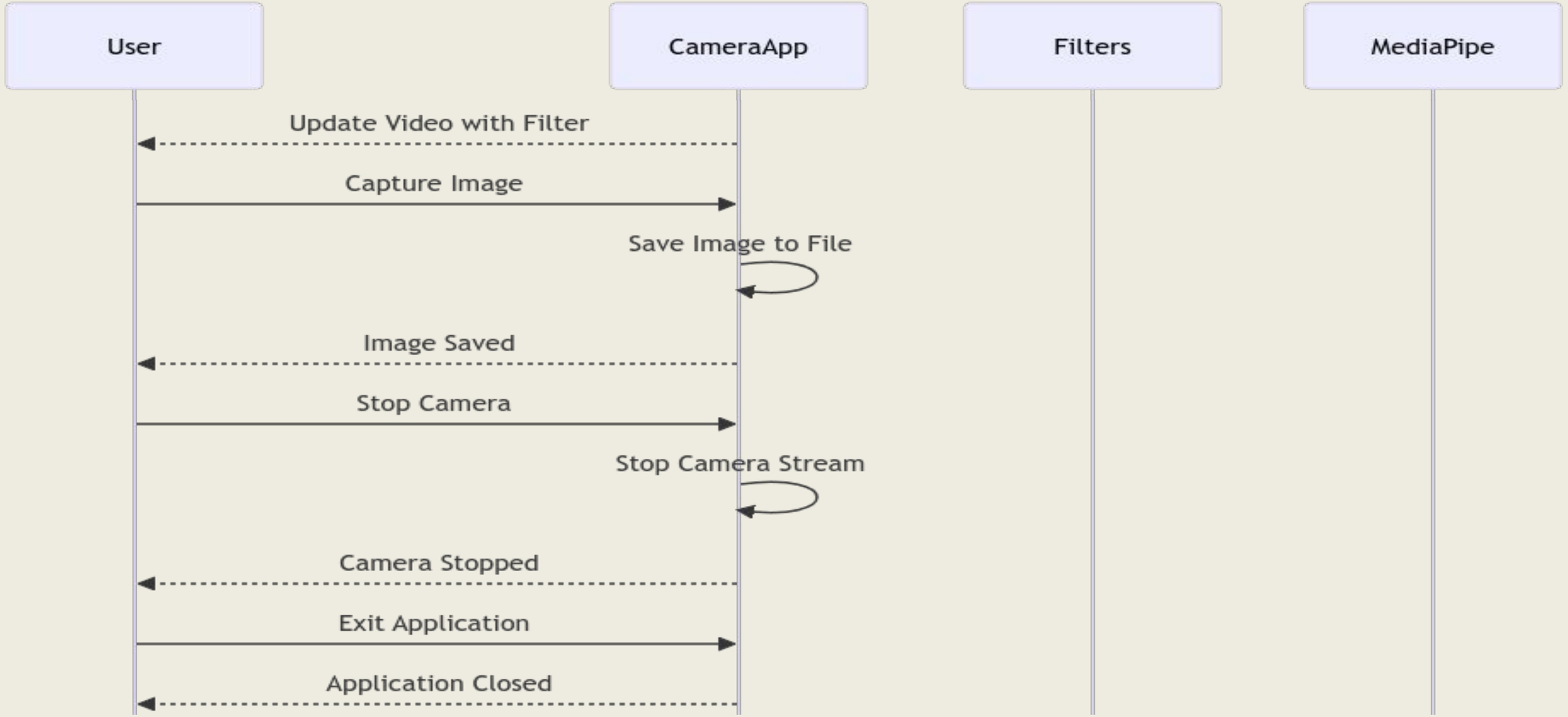


설계 사양서 - Sequence Diagram



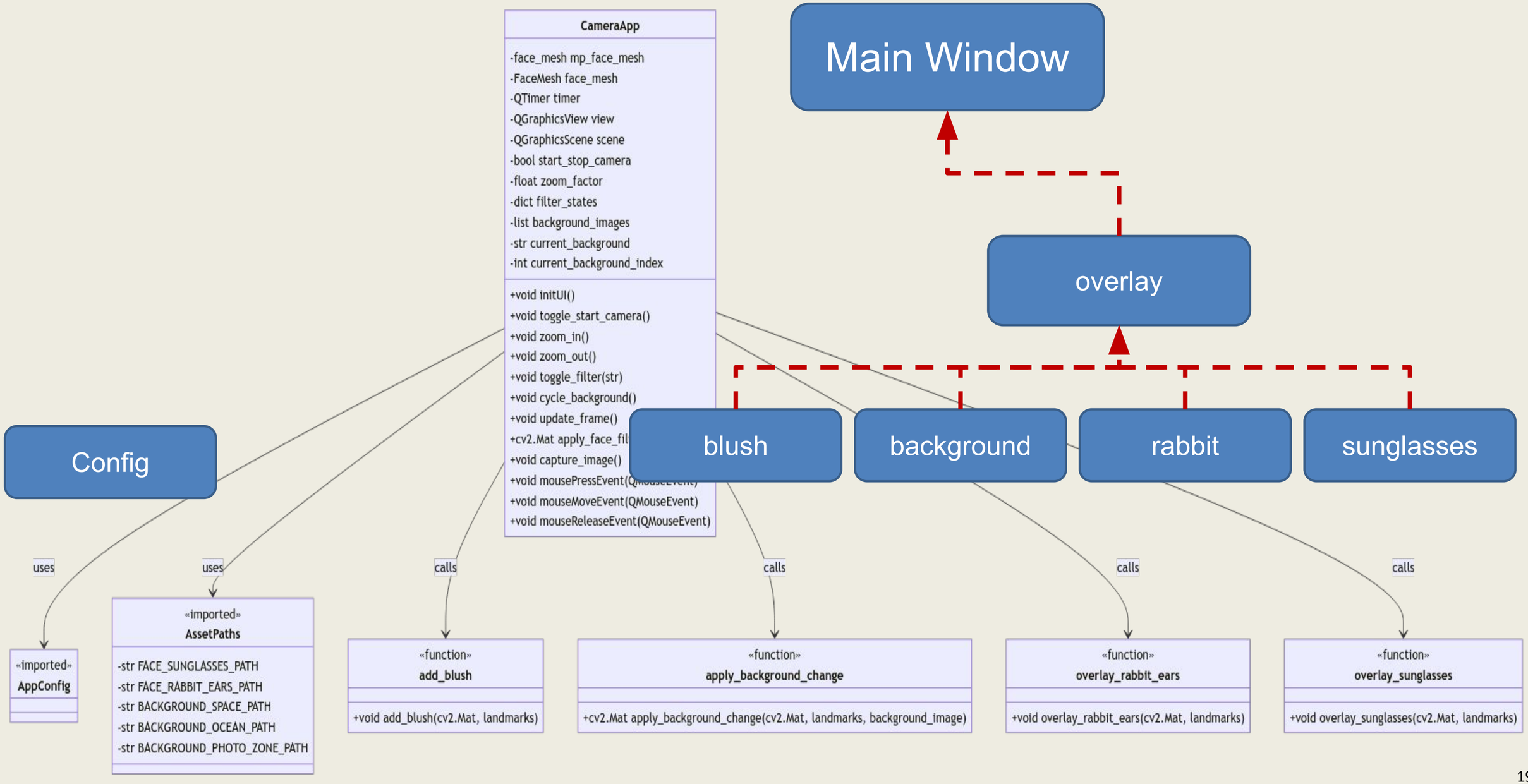
-

설계 사양서 - Sequence Diagram



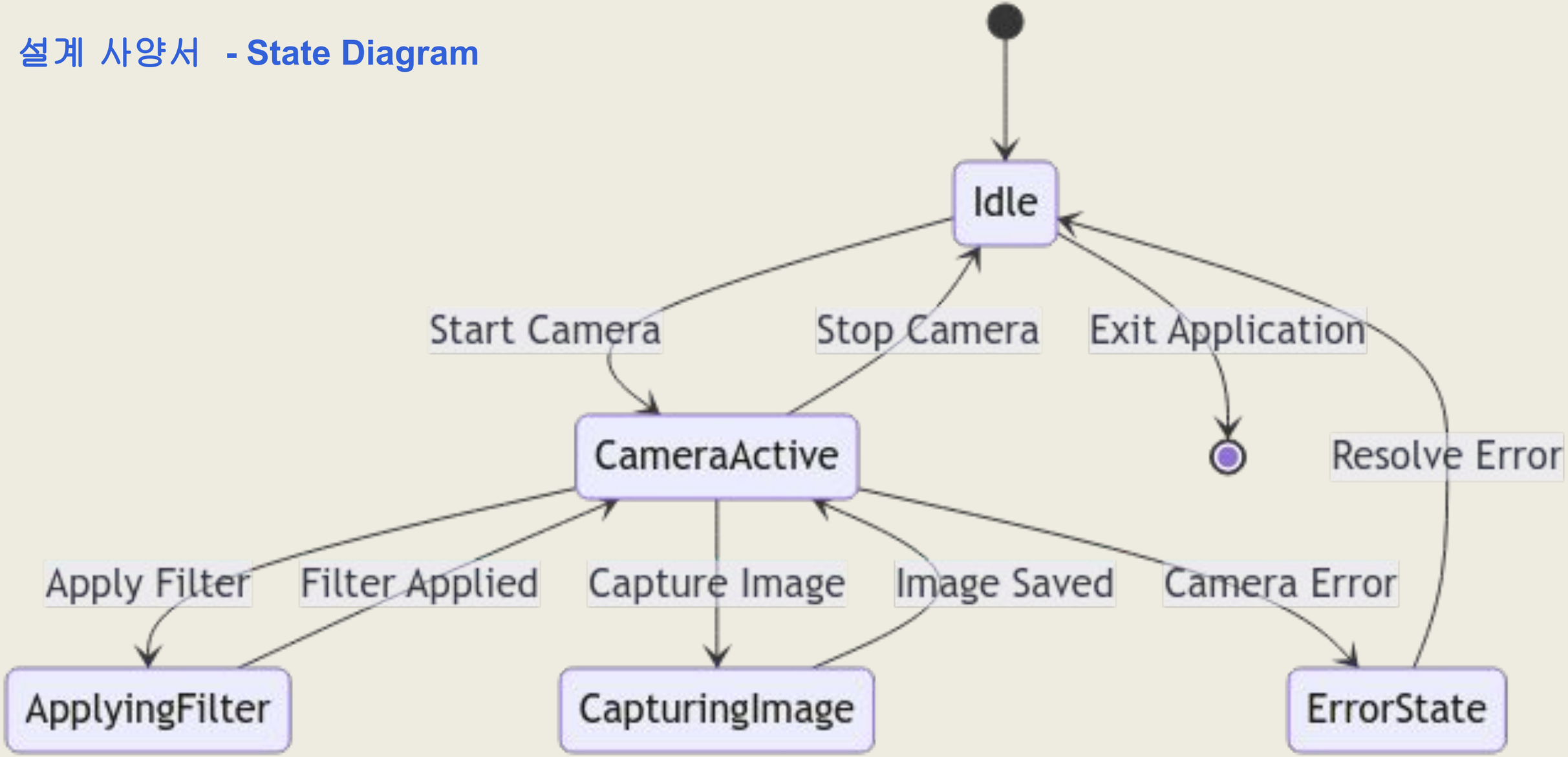
설계 사양서 - Class Diagram

-



설계 사양서 - State Diagram

-



주요 코드

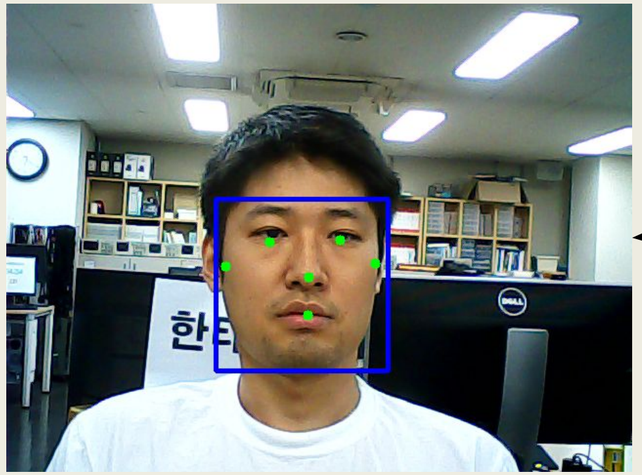
overlay_rabbit_ears()

사진 프레임 받기

얼굴 탐지

얼굴 랜드마크 탐지

함수 호출



overlay_rabbit_ears(frame: cv2.Mat, landmarks: list**)**

주요 코드

overlay_rabbit_ears()

토끼귀 이미지 로드

cv2.imread(AssetPaths.FACE_RABBIT_EARS_PATH, cv2.IMREAD_UNCHANGED)
// 알파 채널 포함해서 읽기

랜드마크 중 “이마” 선택

forehead_index = 10
forehead_landmark =
landmarks[forehead_index]

“이마” 랜드마크의 실제 좌표 계산

forehead_x = int(forehead_landmark.x * w)
forehead_y = int(forehead_landmark.y * h)

주요 코드

overlay_rabbit_ears()

토끼귀 위치 조정

```
x1, y2 = forehead_x - ear_width // 2, forehead_y
x2, y1 = x1 + ear_width, y2 - ear_height
```

토끼귀 크기 재조정

```
x1, y1 = max(0, x1), max(0, y1)
x2, y2 = min(w, x2), min(h, y2)

cropped_width = x2 - x1
cropped_height = y2 - y1
```

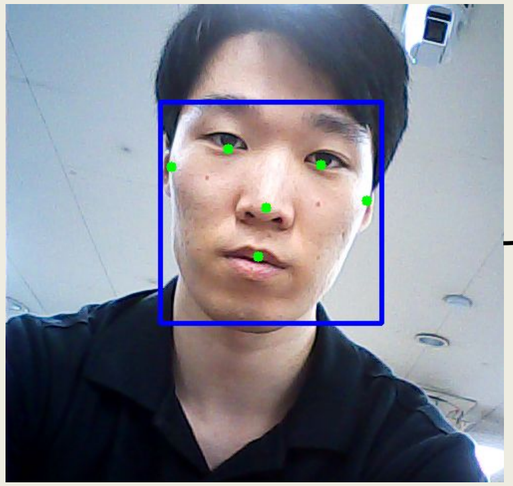
알파 블렌딩 (오버레이)

투명도를 사용한 이미지 합성

```
rabbit_ear_cropped = rabbit_ear_img[-cropped_height:, :cropped_width]
for c in range(3):
    frame[y1:y2, x1:x2, c] = alpha_s * rabbit_ear_cropped[:, :, c] + alpha_l * frame[y1:y2, x1:x2, c] )
```

주요 코드

overlay_sunglasses()

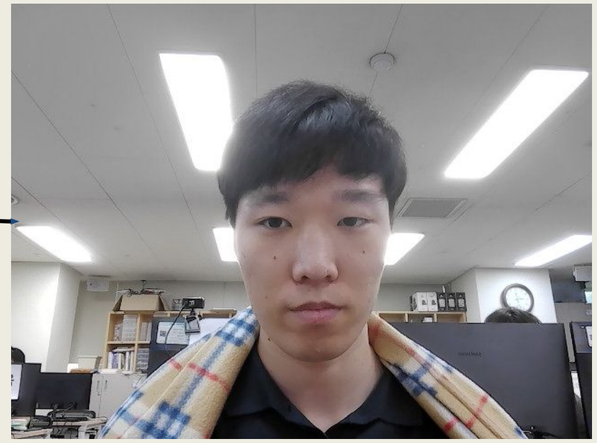


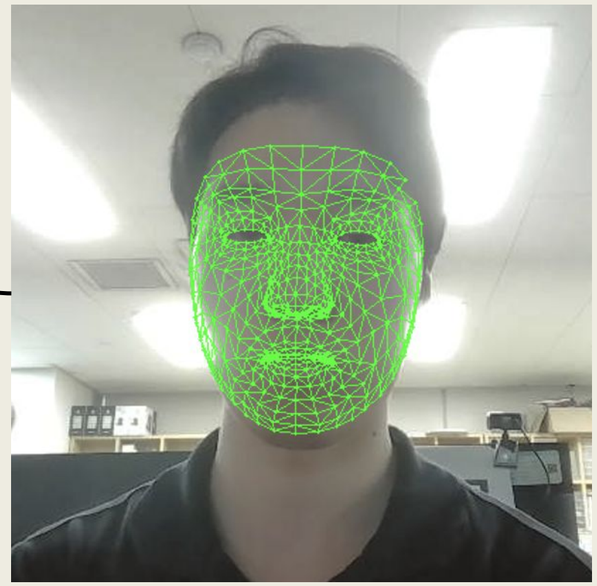
카메라 프레임 받기

얼굴 탐지

얼굴 랜드마크 탐지

함수 호출





overlay_sunglasses(frame, landmarks)

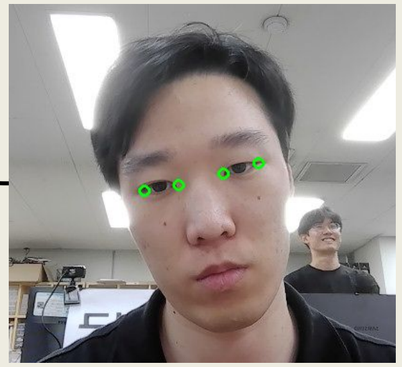
주요 코드

overlay_sunglasses()

선글라스 이미지 로드

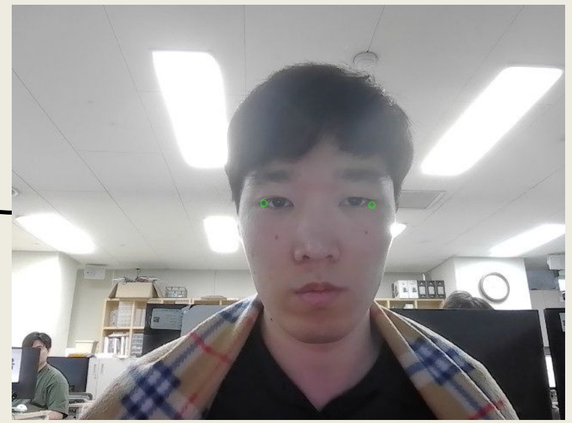
left_eye_indices = [33, 133]
right_eye_indices = [362,263]

눈 index 설정

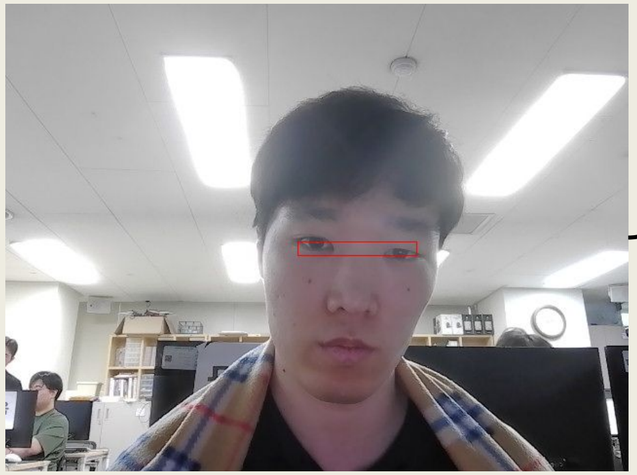


min(), max()를 통해
두 눈의 끝 점을 선택

양 끝점의 좌표 저장



두 점으로 눈의 범위 설정



주요 코드

overlay_sunglasses()

눈 가로/세로 길이 계산

glasses_width = int((x2-x1) * 2.3)
glasses_height = int(0.8 * glasses_width)

선글라스 이미지 Resize

overlay =
cv2.resize(sunglasses_img,(width,height)

주요 코드

overlay_sunglasses()

회전 각도를 계산하고
선글라스 회전

```
angle = calculate_angle(landmarks, frame)
rotated_img = rotate_image(overlay, -angle)
```

cv2.GetRotationMatrix2D()

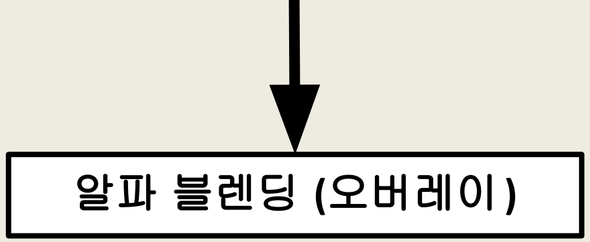
선글라스와 원본 이미지
투명도 계산 후 전달

```
alpha_s = rotated_img[:,:,3] / 255
alpha_l = 1.0 - alpha_s

alpha_s * rotated_img + alpha_l *overlay_img
```

주요 코드

overlay_sunglasses()



```
overlay_img = frame[y1:y2, x1:x2]
for c in range(3):
    overlay_img[:,:,c] = alpha_s * rotated_img[:,:,c] + alpha_l * overlay_img[:,:,c] )
```

capture_image_2
0240904_125947
.png

capture_image_2
0240904_133236
.png

capture_image_2
0240904_133241
.png

capture_image_2
0240904_133244
.png

capture_image_2
0240904_135617
.png

capture_image_2
0240904_135620
.png

capture_image_2
0240904_135622
.png

```
if hasattr(self, “current_frame”):
timestamp = datetime.now().strftime(“%Y%m%d_%H%M%S”)
output_img_path: str
=f”{./Images/capture_image_{timestamp}.png”

cv2.imwrite(output_img_path, self.current_frame)

captured_img = cv2.imread(output_img_path, cv2.IMREAD_COLOR)
cv2.imshow(“Captured Image”, captured_img)
```

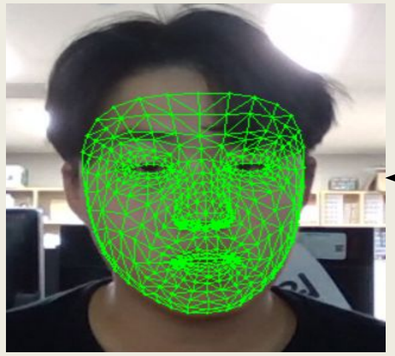
▶ datetime.now()를 통해 현재 날짜와 시간을 문자열로 변환, 저장

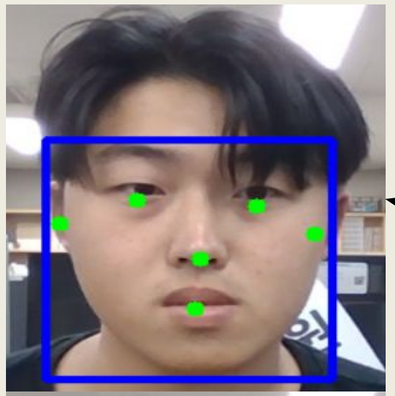
▶ 이 문자열을 저장될 이미지의 이름에 넣어 기존의 파일을 덮어쓰지 않도록 함

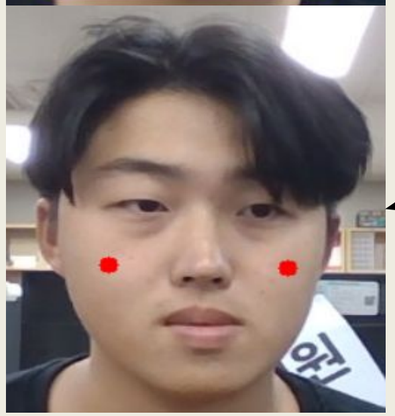
▶ 저장된 사진을 바로 확인할 수 있도록 imshow로 창 띄우기

주요 코드

Blush







이미지 받기

얼굴 영역 좌표 받기

add_blush(frame,landmarks)

양쪽 볼 인덱스 정의

left_check_index = 50
right_check_index = 280

주요 코드

Blush

랜드마크 좌표 생성

픽셀 좌표로 변환

```
landmark = landmarks[index]
(landmark.x*frame.shape[1],
landmark.y*frame.shape[0])
```

픽셀 좌표에 원 생성

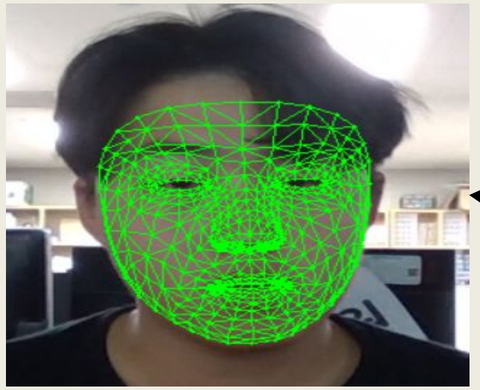
```
cv2.circle(overlay, point,
radius, color, -1)
```

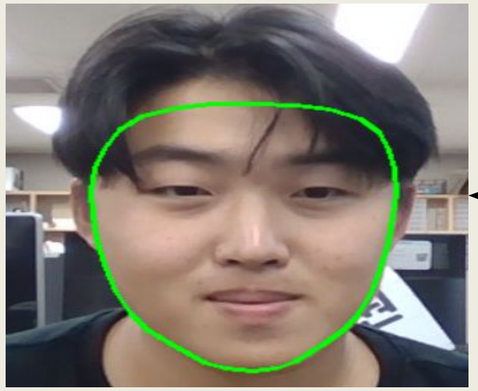
가우시안 필터 적용

```
cv2.GaussianBlur(overlay, (91,91), 0)
```

주요 코드

BackGround





이미지 받기

얼굴 영역 좌표 받기

배경 이미지 받기

얼굴 윤곽 인덱스 배열 생성

apply_background_change(frame, landmarks, background_image)

face_landmarks_indices = [10,338,297,332,284,251,389,356,454,323,361,288,397,365,379,378,400,377,152,148,176,149,150,136,172,58,132,93,234,127,162,21,54,103,67,109,]

주요 코드

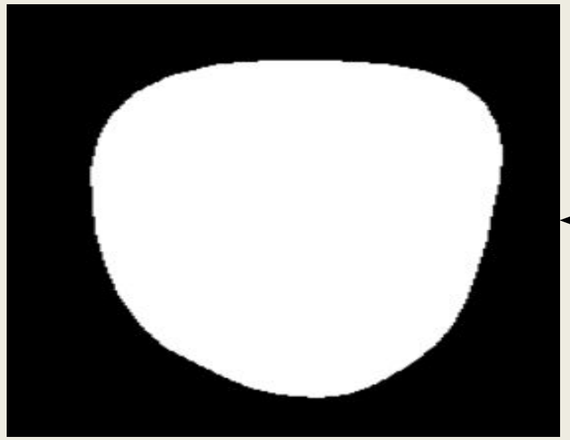
BackGround

픽셀 좌표로 변환

```
points = np.array(
    [
        (
            int(landmarks[idx].x * frame.shape[1]),
            int(landmarks[idx].y * frame.shape[0]),
        )
        for idx in face_landmarks_indices
    ],
    np.int32,
)
```

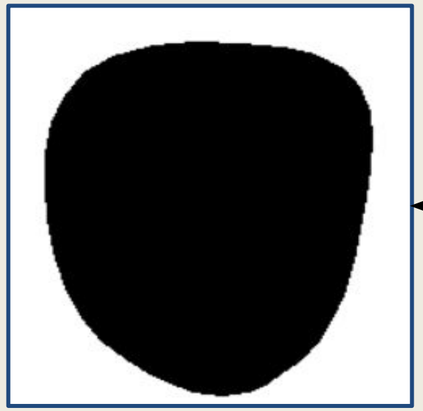
얼굴 픽셀 마스크 정의

```
points = points.reshape((-1, 1, 2))
cv2.fillPoly(mask, [points], (255, 255, 255))
```



주요 코드

BackGround



배경 이미지 resize

```
resize = cv2.resize(background_image,
(frame.shape[1], frame.shape[0]))
```

배경 mask 정의

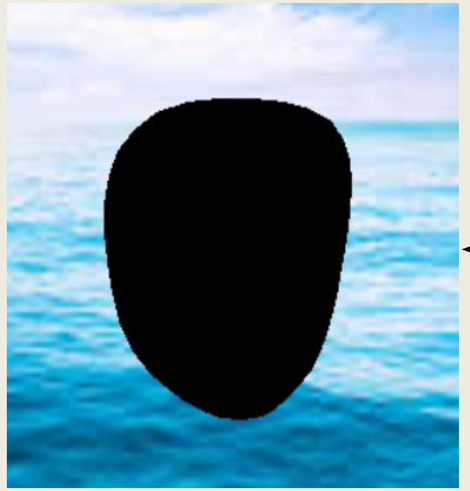
```
inverse_mask = cv2.bitwise_not
(mask)
```

얼굴 픽셀 좌표 원본 적용

```
face_area = cv2.bitwise_and
(frame, frame, mask=mask)
```

주요 코드

BackGround



배경 픽셀 값 대입

```
background_area  =
 cv2.bitwise_and
(resize_background,
resize_background,
mask=inverse_mask)
```

결과물 출력

```
output = cv2.add(face_area, background_area)
```

주요 코드

GUI setting

init UI()호출

1.주요 컴포넌트 초기화

- 중앙 위젯을 생성하고 메인 윈도우의 중앙 위젯으로 설정.
- 메인 윈도우의 크기를 800x600 픽셀로 조정.

2.QgraphicsView 생성
> 2D그래픽 핸들링에 사용

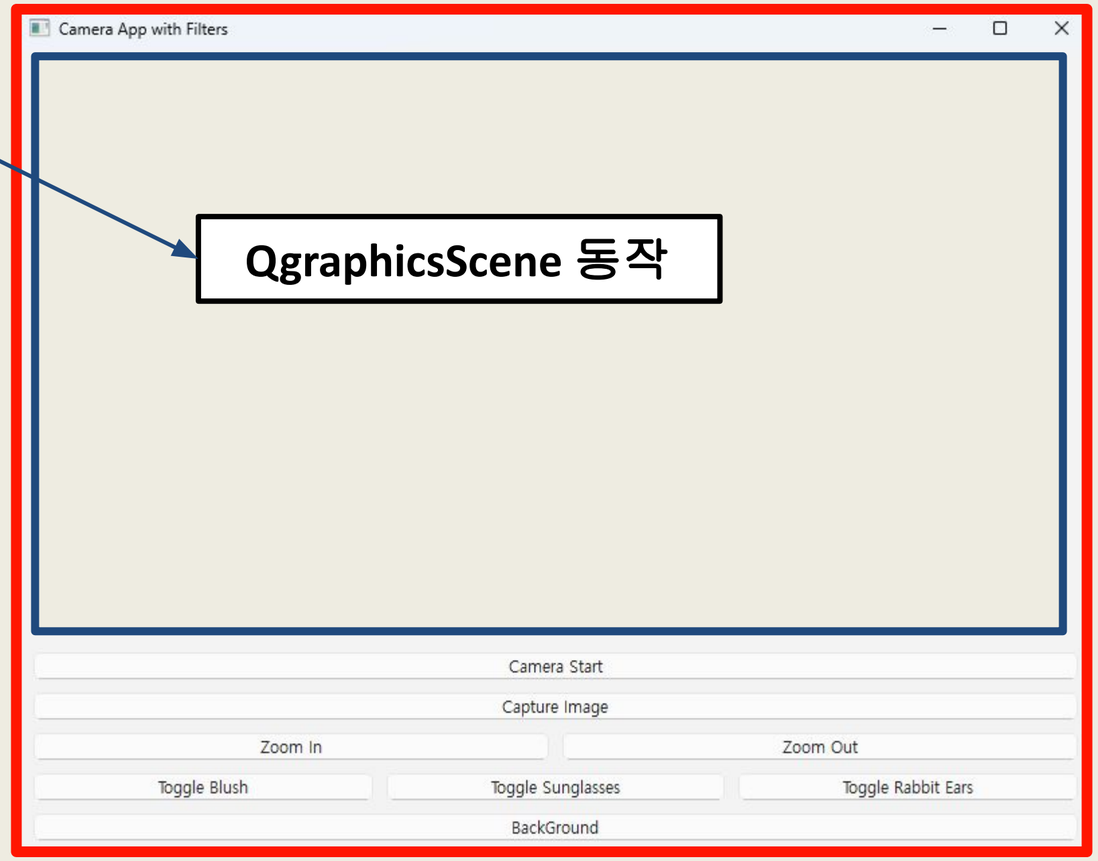
3.Button 생성

Button event 연동

4.layout 설정

Qvboxleyout() : 수직

QgraphicsScene 동작



Qhboxleyout() : 수평

부가 기능
- 카카오톡 연동

개요

대한 민국 국민이 가장 많이 사용하는 메신저인 **‘카카오톡’**에 찍은 사진을 바로 공유하는 기능을 구현해보고자 하였다.

주요기능

[인텔 미니 프로젝트] 채널 추가만 하면 앱으로 찍은 최근 사진을 바로 불러오기 가능

부가 기능
- 사용한 오픈소스

flask

라우팅 (특정 URL 접속시 특정 동작 수행 정의), 요청 및 응답 처리, 세션 관리등

웹 제작 최소 기능 제공 라이브러리

ngrok

로컬 서버를 외부에서 접근할 수 있는 URL 제공

마침 접근이 허용된 api가 있는 계정이 있기에 시도하였다.

kakao openbuilder

카카오톡 생성 채널과 ngrok 생성 웹페이지를 연동할 챗봇 서비스 제공

부가 기능

- **kakao open builder**

[시나리오]
블록, 이라 불리는 fx 를 추가할 수 있다.

1. 웰컴 블록 : 사용자가 채널에 처음 진입했을시 실행되는 블록
2. 폴백 블록 : 학습되지 않은 발화가 입력시 호출되는 블록
3. 탈출 블록 : 챗봇의 사용을 중지하고 싶을시 호출되는 블록

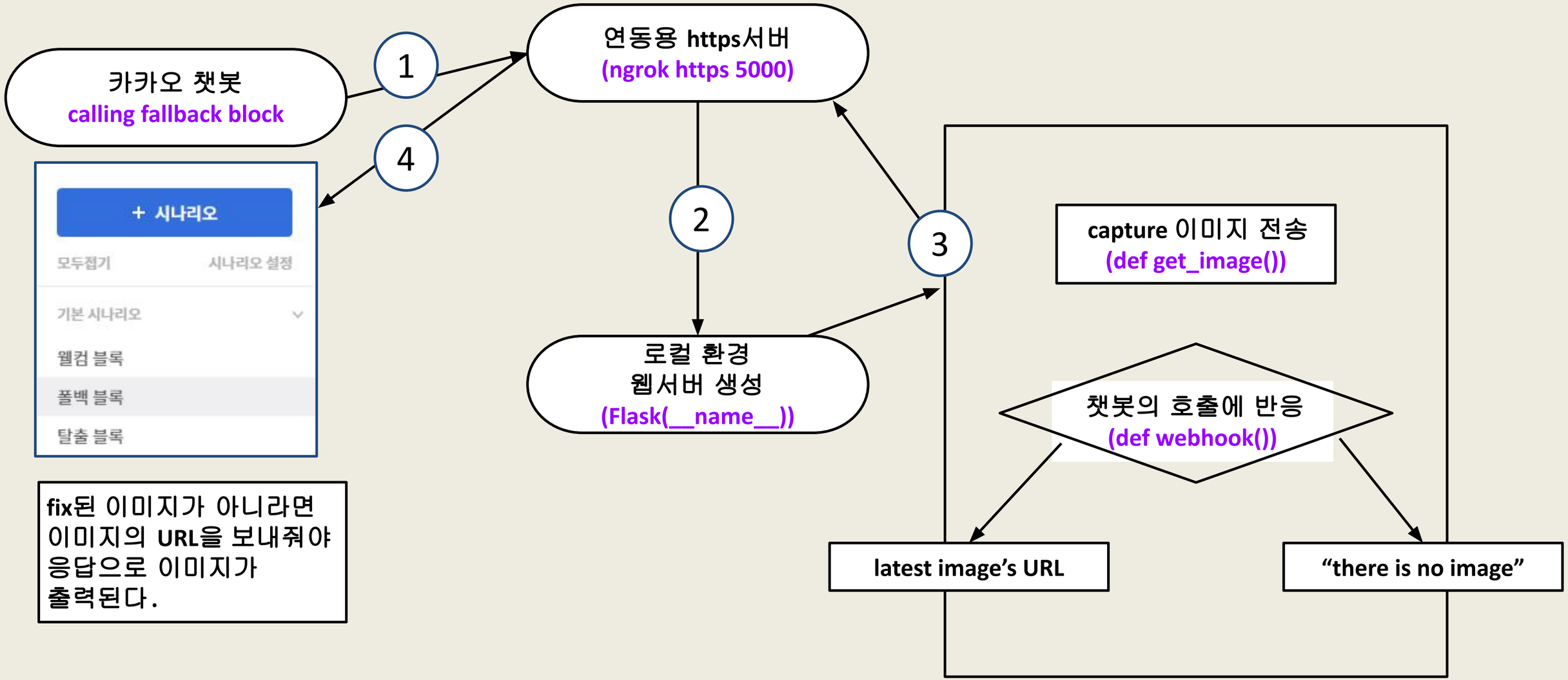
[IDEA]

학습 시킨것이 없는 빈 챗봇모델을 사용하여 무조건 폴백 블록이 호출되게 한다.

응답으로 **skill data**를 사용하여 어떤 질문에도 이미지를 전송하게 한다.

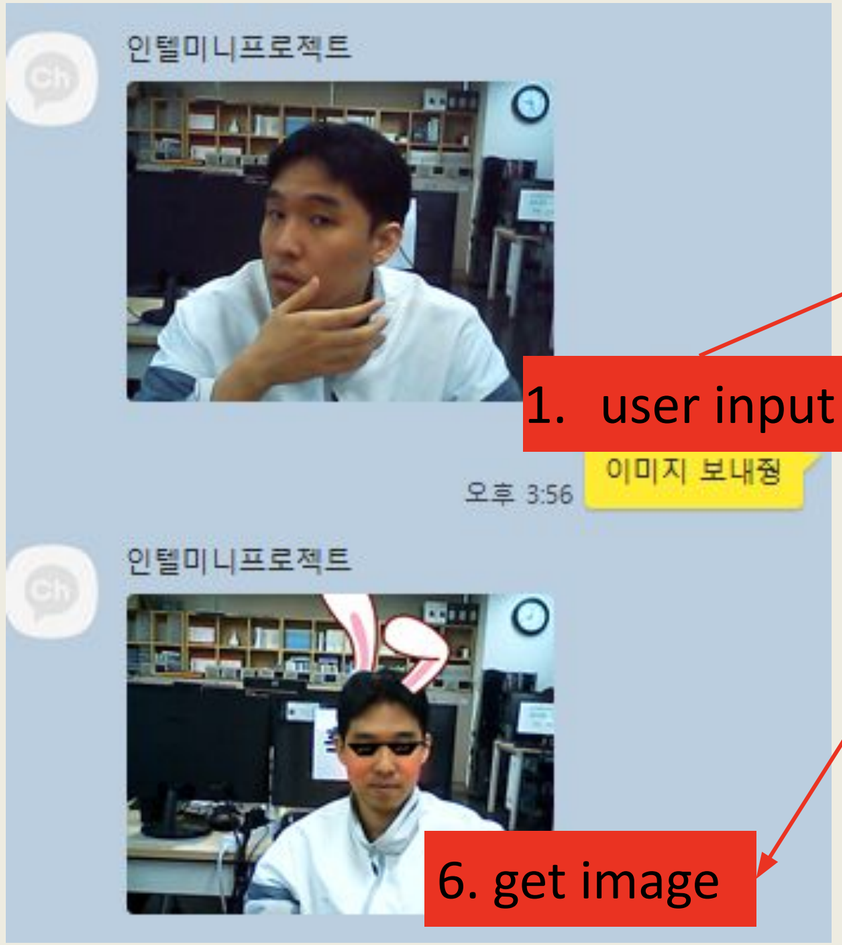
부가 기능

- skill data



부가 기능
- 실행 화면

kakao



1. user input

6. get image

ngrok(https)

```
HTTP Requests
-------------

18:10:34.131 KST POST /webhook                                  200 OK
18:10:34.619 KST GET  /image/capture_image_20240904_131405.png  200 OK
15:46:55.875 KST GET  /image/capture_image_20240904_131405.png  200 OK
15:46:55.494 KST POST /webhook                                  200 OK
15:46:28.439 KST GET  /image/capture_image_20240904_131405.png  404 NOT FOUND
15:46:28.101 KST POST /webhook                                  200 OK
15:46:28.807 KST GET  /image/capture_image_20240904_131405.png  404 NOT FOUND
13:11:18.039 KST GET  /image/capture_image_20240904_131110.png  200 OK
13:11:17.845 KST POST /webhook                                  200 OK
13:1                  ebhook                                    200 OK
```

2. call webhook()

5. get image

flask(local)

```
User input: 이미지 보내줭
127.0.0.1 - - [04/Sep/2024 18:32:32] "POST /webhook HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2024 18:32:32] "GET /image/capture_image_20240904_131405.png HTTP/1.1" 200 -
```

3. get req
4. send image

부가 기능
- 한계점

로컬 서버를 사용하는 코드의 한계

로컬 서버를 억지로 https url을 부여하여 연결중이기에 특정 기기의 이미지만 출력가능하다.

=> 해결법 : (naver cloud) 등 고정된 IP와 공유 서버를 제공하는 서비스를 이용한다.

가장 최근 파일만 제공하는 코드의 한계

sorted 함수로 이미지를 정렬 후 가장 마지막 이미지를 보내주는 방식으로 동작 중 이기에 최근 사진이 아닌 사진은 받아 볼 수 없다.

=> 해결법 : 카카오톡 빌더에 블록을 추가하여 사용자가 필요한 사진이 몇번째 사진인지 물어보는 과정을 추가해준다.

느낀점

필터 추가, 부가 기능 추가하는 과정에서 객체화와 리팩토링의 중요성을 느낄 수 있었습니다.

한태섭

MediaPipe 를 쓸 때 **Legacy API** 를 사용하여 파이썬에서 타입 힌트를 받지 못하여, 코드를 자유롭게 작성하고 테스트해볼 때 어려움이 있었다.

박준수

조원들과 코드를 통합하여 함께 개발하다 보니 코드의 가독성과 변수명 설정이 개발과정에 있어서 중요하다는 것을 깨달았습니다.

권시우

특정 사진을 필터로 만들어 오버레이 하는 과정에서 크기, 각도,알파블렌딩 등을 고려하면서 어려움이 있었다.

최재원

2조

THE END