영상처리프로그래밍

기말프로젝트 보고서

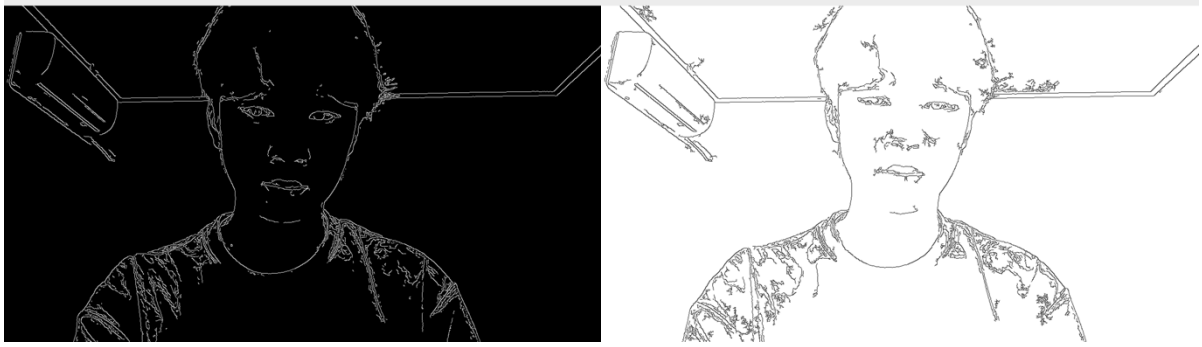
20185109 길홍배

1. 기능

캠을 통해 이미지를 받아와 컨볼루션, 기하학처리, 행렬 연산, 색상 공간 변경 등 수업시간에 배운 내용들을 응용하여 틱톡, 인터넷 방송 캠 처럼 여러가지 이미지 변환을 해주는 프로그램 입니다. 프로그램에 포함되어 있는 기능들은 크게 두가지로 나뉩니다. 첫번째는 기하학 처리를 통해 캠을 2, 4, 9분할 해주는 기능이 포함되어 있습니다. 두번째는 컨볼루션, 행렬 연산, 색상 변경, 이미지 딜레이와 같은 평범한 이미지에 특수한 효과를 주는 기능들이 포함되어 있습니다. 방향키를 눌러 카메라 모드와, 필터를 변경 할 수 있도록 만들었습니다.

HSV를 이용하여 보색으로 나타내기

케니 에지검출을 이용한 필터



딜레이 스크린 구현

2. 코드 설명

파일은 프로그램의 시작인 main.py, 필터와 카메라 모드 적용을 위한 클래스와 함수가 구현 되어 있는 util.py, 카메라 분할모드를 적용하는 mode.py, 필터를 적용하는 filter.py로 구성되어 있습니다. 구현되어 있는 기능들은 모두 강의자료에 있는 내장함수나 기능들을 직접 구현하여 사용하였습니다.

main.py

메인 파일은 이미지를 프레임 단위로 받아와 출력하는 기본 코드로 되어 있습니다.

```
capture = cv2.VideoCapture(0)

if capture.isOpened() is False:
    raise Exception("카메라 연결 안됨")

camera_set = utils.CameraSet()

while True:
    ret, frame = capture.read()
    if not ret: break

    key = cv2.waitKeyEx(1)
    if key == 27: break
    else: key_input(key, camera_set)

    frame = get_filtered_image(frame, camera_set)
    frame = get_divided_image(frame, camera_set)

    cv2.imshow("camera", frame)
capture.release()
```

util.py

util파일에는 카메라 필터와 분할을 위한 열거형과, 딜레이 스크린을 위한 클래스, 키 입력을 받으면 카메라와 필터 모드를 변경해주는 함수가 작성되어있습니다.

```python
class DelayScreen:
    def __init__(self):
        self.count = 0
        self.delay_image = [None for i in range(90)]
        self.delay = 10

    def increase_count(self):
        self.count += 1

    def set_delay_image(self, image, i):
        self.delay_image[i] = image

    def set_count(self, count):
        self.count = count

    def set_delay(self, delay):
        self.delay = delay

    def get_delay_image(self, i):
        return self.delay_image[i]

    def get_count(self):
        return self.count

    def get_delay(self):
        return self.delay
```

```python
class CameraSet:
    def __init__(self):
        self.div_mode = 1
        self.filter_mode = 1

    def set_div_mode(self, div_mode):
        self.div_mode = div_mode

    def set_filter_mode(self, filter_mode):
        self.filter_mode = filter_mode

    def get_div_mode(self):
        return self.div_mode

    def get_filter_mode(self):
        return self.filter_mode
```

```python
class Division(Enum):
    NORMAL = 1
    DIVISION_2 = 2
    REV_DIVISION_2 = 3
    DIVISION_3 = 4
    DIVISION_4 = 5
    REV_DIVISION_4 = 6
    DIVISION_9 = 7
    SIZE = 8
def key_input(key, camera_set):
    filter_mode = camera_set.get_filter_mode()
    div_mode = camera_set.get_div_mode()

    if key == 63232:
        camera_set.set_filter_mode(filter_next(filter_mode))
    if key == 63233:
        camera_set.set_filter_mode(filter_previous(filter_mode))
    if key == 63234:
        camera_set.set_div_mode(division_previous(div_mode))
    if key == 63235:
        camera_set.set_div_mode(division_next(div_mode))

    return div_mode, filter_mode
```

```python
def get_divided_image(image, camera_set):
    dst = np.array(image)
    div_mode = camera_set.get_div_mode()

    if div_mode is Division.DIVISION_2.value:
        dst = division2(image)
    elif div_mode is Division.REV_DIVISION_2.value:
        dst = reverse_division2(image)
    elif div_mode is Division.DIVISION_3.value:
        dst = division3(image)
    elif div_mode is Division.DIVISION_4.value:
        dst = division4(image)
    elif div_mode is Division.REV_DIVISION_4.value:
        dst = reverse_division4(image)
    elif div_mode is Division.DIVISION_9.value:
        dst = division9(image)

    return dst
```

mode.py

화면을 여러 개로 분할 해주는 함수들이 작성되어 있습니다. 기하학 변환을 이용해 함수의 기능들을 구현하였습니다. Scaling 함수를 작성하였지만 성능을 높이기 위해 내장함수인 cv2.resize를 사용하였습니다. Scaling 함수는 제대로 작동하는 것을 확인하였습니다.

화면을 2분할 하는 함수

```python
def division2(image):
    dst = np.zeros_like(image)
    # size = (int(dst.shape[0]), int(dst.shape[1] * 0.5), 3)
    # dst_2 = scaling(image, size)
    size = (int(dst.shape[1] * 0.5), int(dst.shape[0]))
    dst_2 = cv2.resize(image, size, 0, 0, cv2.INTER_LINEAR)

    dst = cv2.repeat(dst_2, 1, 2)

    # d2_size = dst_2.shape[1]
    # dst[:, :dst_2.shape[1]] = dst_2
    # dst[:, dst_2.shape[1]:image.shape[1]] = dst_2
    #
    return dst
```

```python
# def scaling(image, size):
#     dst = np.zeros(size, np.uint8)
#     ratio_y, ratio_x = image.shape[0] / size[0], image.shape[1] / size[1]
#     for y in range(image.shape[0]):
#         for x in range(image.shape[1]):
#             i, j = int(y / ratio_y), int(x / ratio_x)
#             dst[i, j, ::] = image[y, x, ::]
#
#     return dst
```

화면을 2분할 하는 것 뿐만 아니라 4, 9, 좌우반전 등 여러가지의 기능이 구현되어 있습니다.

filter.py

카메라 필터를 적용하는 함수들이 구현되어 있는 파일입니다. 모든 기능은 강의자료를 참고하여 각각의 함수로 작성되어 있습니다. 그 중 프로젝트에서 가장 큰 기능을 담당하는 기능은 딜레이 스크린 입니다.

HSV를 이용한 보색

이미지를 픽셀이미지처럼 변환

```python
def complementary_image(image):
    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv_img)
    temp = (h + 90) % 181
    h = np.array(temp, np.uint8)
    c_hsv = cv2.merge((h, s, v))
    dst = cv2.cvtColor(c_hsv, cv2.COLOR_HSV2BGR)

    return dst
```

```python
def image_to_pixel(image):
    # dst_size = (int(dst.shape[0] * 0.05), int(dst.shape[1] * 0.05), 3)
    # dst = scaling(image, size)
    dst_size = (int(image.shape[1] * 0.05), int(image.shape[0] * 0.05))
    dst = cv2.resize(image, dst_size, interpolation=cv2.INTER_LINEAR)
    # img_size = (int(image.shape[0]), int(image.shape[1]), 3)
    # pixel = scaling(image, size)
    img_size = (image.shape[1], image.shape[0])
    pixel = cv2.resize(dst, img_size, interpolation=cv2.INTER_LINEAR)
    return pixel
```

딜레이 스크린

```python
def delay_screen(image, divide=9):
    dst = np.zeros_like(image, np.uint8)
    dx, dy = (int(image.shape[1] / 3), int(image.shape[0] / 3))
    divide_img = cv2.resize(image, (dx, dy), 0, 0, cv2.INTER_LINEAR)

    count = delay.get_count()
    delay_time = delay.get_delay()

    if count == sys.maxsize:
        delay.set_count(0)
```

```python
    for i in range(divide):
        index = i * delay_time
        width = i % 3
        height = int(i / 3)
        delay.set_delay_image(divide_img, delay.count % (delay_time * divide))

        if count >= index:
            delay_img = delay.get_delay_image((count - index) % (delay_time * divide))
            dst[dy * height: dy * (height + 1), dx * width: dx * (width + 1)] = delay_img

    delay.increase_count()

    return dst
```

해당 코드를 설명하자면 util.py에 선언된 DelayScreen 클래스를 사용합니다. 이미지를 9분할 하기 위해 변환을 하여 divide_img에 저장합니다. 이후 프레임 단위로 카운트를 셉니다. 이미지가 9분할 되었으니 한 이미지당 10프레임의 딜레이를 주어 9번째 이미지는 90프레임의 딜레이가 생기게 만들 것 입니다. 카운트를 프레임으로 이용할 것입니다. 따라서 각 프레임마다 클래스의 delay_image 리스트에 저장합니다. 그렇게되면 총 90프레임의 이미지가 저장이 되게 됩니다. 이미지를 프레임별로 저장함과 동시에 10프레임 단위마다 순서대로 이미지를 출력해 줄 것입니다. 카운트가 20이라고 치면 세번째 이미지에 0번째 프레임의 이미지를 받아와야 하는 것 입니다. 따라서 if문 안에 있는 수식을 사용하여 이미지에 딜레이를 주어 화면에 출력합니다.

3. 아쉬운 점

처음 기획했던것 보다 필터가 부족한 것이 아쉽습니다. 현재 구현한 기능들 이외에도 인스타그램의 릴스나 틱톡 같은 필터를 구현하고 싶었습니다. 예를 들면 이미지 위에서 바가 내려오는데 그 바를 지나는 이미지들은 고정이 되고 바가 끝까지 내려오는 그런 기능을 만들고 싶었으나 구현하지 못했고, 구현된 기능 중 AfterImage 라는 기능이 있습니다. 해당 기능은 딜레이 스크린과 비슷하게 한 화면에서 이미지가 딜레이 되어 덮어 씌여지는 기능을 구현하고 싶었으나 해당 기능은 움직이는 물체의 테두리를 따서 100%의 투명도로 덮어씌워야 하는 기능이 필요할 것 같아 구현하지 못했습니다.