2023/01/02-08

연구제목 : 영상 처리 성능 향상

연구목적 : 영상 인식 및 판단 성능 향상과 리소스 사용량 감소

작성주차 : 1월 1주차

작성자명 : 서동환

에러노트 : GPU 사용량 과다, 리눅스 및 환경설정, 빛 번짐

세부사항 :

1. 연구 주제 : GPU 사용량 감소

선택 이유 : 지금은 CCTV의 모든 frame을 yolo를 거치기 때문에 GPU가 사용되지 않는 시간이 없다. 따라서 GPU의 사용 시간을 줄이기 위해 화면에서 움직임이 검출되면 그 때 부터 yolo를 적용시키는 방안에 대해 연구 해 보았다.

과정:

1안. 움직임 검출

→ 화면에 아무런 변화가 없을 때(움직이는 물체가 없을 때)에는 굳이 yolo가 화면에서 객체 검출을 할 필요가 없다. 따라서 화면에 변화가 감지되면 그 때의 frame부터 yolo가 객체 탐지를 하고 변화가 5분이상 없을 시에는 다시 yolo의 활동을 멈추면 된다.

- CodeFlow

  차영상으로 움직임 검출 →

  움직임이 검출되면 그때부터 원본 frame을 yolo에 넘겨주기 →

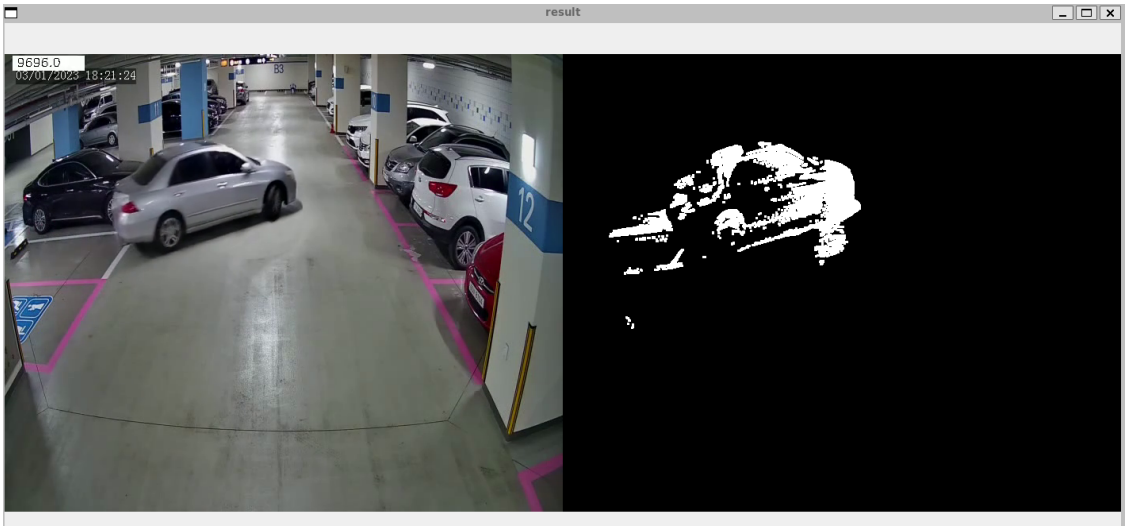
  5분 이상 움직임이 없다면 frame 전달 중단

현재 구현 상황 :

- 차영상으로 움직임 검출(구현 완료) :

  차영상은 크게 KNN, MOG2로 두가지 방법이 있는데, 여러번 테스트 해본 결과 KNN이 압도적으로 노이즈가 적었기 때문에 KNN(cv2.createBackgroundSubtractorKNN)를 선택해서 차영상을 추출하였다. 근데 그냥 영상을 KNN으로 하면 KNN이라도 노이즈가 많기 때문에 GRAY처리를 하고 KNN에 적용해 주었다. 나머지 하이퍼 파라미터들은 좀 더 다양한 방식(blur처리, OPEN or CLOSE 등)으로 테스트를 해 봐야 할 것 같다. 마지막으로 움직이고 있는 픽셀 개수를 구해줌으로서 픽셀 개수가 일정 개수를 넘기면 움직이는 물체가 있다 라고 판단하게 했다.

  -아래 그림은 영상 일부분 캡쳐



- 움직임이 검출되면 그때부터 원본 frame을 yolo에 넘겨주기 (구현 완료):

  yolo로 frame을 넘겨주는 법을 몰라 이 부분은 그냥 영상을 녹화 하는 코드(주석처리상태)로 구현해 놓았다.

- 5분 이상 움직임이 없다면 frame 전달 중단(구현완료) :

  위에서 구한 변하는 픽셀 개수가 일정량을 넘으면 녹화가 시작되며, 움직임이 없는 상태로 약 2분이 지속되면 녹화가 종료되도록 해두었다.

▼ 코드

```
import cv2
import time
import numpy as np

#불러올 영상 & 녹화할 영상 위치
capture = cv2.VideoCapture("./CCTV17_crop_2.mp4")
output_path = './output_result.mp4'

#불러온 영상의 가로,세로,fps & 녹화할 영상 저장 형식
width = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(capture.get(cv2.CAP_PROP_FPS))
codec = cv2.VideoWriter_fourcc(*'mp4v')

# 배경 추출을 위한 KNN,kernel 설정
bget_KNN = cv2.createBackgroundSubtractorKNN(detectShadows=False)

kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))

#init
time_cost = 0
fps_cost = 0
frame_num = 0
check = 0
record = False

# 영상 frame 별로 처리
while True:
    key = cv2.waitKey(33)
    start = time.time()
    frame_num +=1

    #ESC 누르면 종료됨
    if key == 27:
        break

    return_value, frame = capture.read()
    # 비디오 프레임 정보가 있으면 계속 진행
    if return_value:
```

```
        pass
    else :
        print('비디오가 끝났거나 오류가 있습니다')
        break

    # Gray로 바꿔주고 noise 제거를 위한 blur 처리 -> blur 처리가 오히려 객체 탐지에 방해되서 주석 처리
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #frame_gray = cv2.GaussianBlur(frame_gray,(3,3),0)

    # 배경 추출을 위한 KNN 적용
    backgroundsub_mask_KNN = bget_KNN.apply(frame_gray)
    backgroundsub_mask_KNN = cv2.morphologyEx(backgroundsub_mask_KNN,cv2.MORPH_OPEN, kernel)
    #backgroundsub_mask_KNN = cv2.morphologyEx(backgroundsub_mask_KNN,cv2.MORPH_CLOSE, kernel)
    backgroundsub_mask_KNN = np.stack((backgroundsub_mask_KNN,)*3, axis=-1)
    #bitwise_image_KNN = cv2.bitwise_and(frame, backgroundsub_mask_KNN)

    # 변화 정도를 구하기
    for_diff = cv2.cvtColor(backgroundsub_mask_KNN, cv2.COLOR_BGR2GRAY)
    diff_cnt = cv2.countNonZero(for_diff)
    #print(diff_cnt)
    #time.sleep(1)

    # 변화가 있으면 frame 전달 시작, 일정 시간 동안 변화 없으면 전달 중지
    # 첫 frame_num > 5 는 초반에 변화 정도가 급격하게 커서 급한대로 추가 -> 수정 필요
    # 기본적인 순환복잡도가 너무 높음 -> 수정 필요
    if frame_num > 5:
        if(diff_cnt > 1000):
            if record == True:
                print("frame 전달중")
                check += 1
                #video.write(frame)
            else:
                print('frame 전달 시작')
                record = True
                check += 1
                #video = cv2.VideoWriter('output_result.mp4',codec, 20.0, (frame.shape[1], frame.shape[0]))
        else:
            if record == True:
                if check < 10000: # 시간조건
                    print("움직임 없지만 frame 전달")
                    check += 1
                    #video.write(frame)
                else:
                    print("frame 전달 종료")
                    record = False
                    check = 0
                    #video.release()
            else:
                print("움직임 없어서 frame 전달 안함")
                pass

    #소요 시간 및 평균 FPS 구하기
    time_temp = 1000*(time.time()-start)
    if time.time()-start != 0.0:
        fps_temp = 1/(time.time()-start)
    else:
        fps_temp=0.01

    time_cost = time_cost + time_temp
    fps_cost = fps_cost + fps_temp
    #print('소요 시간 : {:.2f} ms \t 평균FPS : {:.2f}'.format(time_temp,fps_temp))

    #결과창 봉합해서 출력
    concat_image = np.concatenate((frame,backgroundsub_mask_KNN), axis=1)

    cv2. imshow('result', concat_image)
    #cv2.imshow('backgroundsub_KNN', backgroundsub_mask_KNN)
    #cv2.imshow('bitwise_KNN', bitwise_image_KNN)
    #cv2.imshow('original', frame)


#메모리 반납
print('소요시간 평균 : {:.2f} ms\t 평균FPS : {:.2f}'.format(time_cost / frame_num, fps_cost/ frame_num))
capture.release()
cv2.destroyAllWindows()
```

더 연구해야 할 점 :

1. 함수형 설계

   현재 구현한 코드를 함수로서 받는 인자와 return 값을 정확히 명시해 두지 않으면 프로젝트에 절대 올리 수 없다고 느꼈다. 좀 더 공부해와서 함수로 표현하는게 익숙하고 정확해 져야 겠다라고 느꼈다.

2. 차량의 움직임 확인

   위의 코드는 픽셀 변화에 따른 움직임 자체를 판단하고 있다 보니 화면 밖에서 차량의 라이트가 해당 화면을 비추면 움직임이 있다고 판단하고 영상을 yolo에 보낸다. 따라서 픽셀의 변화에 따른 판단 보다는 객체의 움직임에 따른 판단을 하는게 더 정확할 것이기에, 이 방법에 대해서 좀 더 연구해 볼 필요가 있다.

3. 사람

   영상 멀리서 사람이 움직일 때는 변화가 크지 않기 때문에 변화가 없다고 판단한다. 지금 당장은 차량만 신경쓰고 있지만, 결국에는 사람이 움직이는 것도 파악해야 하기 때문에 알고리즘의 변화가 필요할 것 같다.(사람의 움직임을 판단하고자 임계점을 내려버리면, 단순 noise도 변화라고 인식)

4. 순환 복잡도

   if문 안에 if가 너무 많다. 어떻게 해야 순환복잡도가 줄어들까 고민해봐야 한다.

2. 연구 주제 : 빛 반사로 인한 객체 미인식 완화

선택 이유 : 지하주차장은 야외 주차장과는 다르게 일조량의 변화에 따른 RGB값의 변화가 없을 줄 알았는데, 차량의 라이트가 계속해서 객체 파악에 방해를 야기한다. 따라서 화면이 빛에 인해 변하는 값이 둔화되도록 필터를 설정해야 겠다 판단하였다.

과정 :

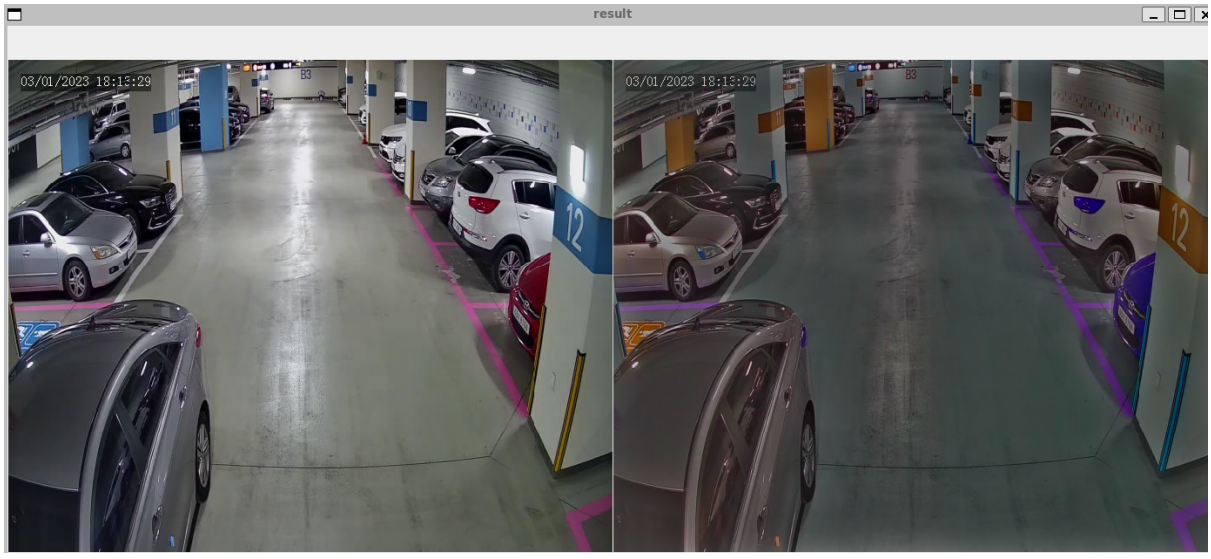
1안. Homomorphic Filter 적용

Homomorphic Filter는 복잡한 수식으로 이루어져 있는데 일단 자세한 방식은 나중에 공부하고 일단 지금은 적용만 해보았다. 하지만 Homomorphic Filter 는 opencv에서 구현되어있는게 아니다 보니 직접 수학 식을 만들어 내야 했다.

▼ 코드

```
import cv2
import numpy as np

### YUV color space로 converting한 뒤 Y에 대해 연산을 진행
img = cv2.imread('CCTV_test.png')
img_YUV = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
y = img_YUV[:,:,0]

rows = y.shape[0]
cols = y.shape[1]

### illumination elements와 reflectance elements를 분리하기 위해 log를 취함
imgLog = np.log1p(np.array(y, dtype='float') / 255) # y값을 0~1사이로 조정한 뒤 log(x+1)

### frequency를 이미지로 나타내면 4분면에 대칭적으로 나타나므로
### 4분면 중 하나에 이미지를 대응시키기 위해 row와 column을 2배씩 늘려줌
M = 2*rows + 1
N = 2*cols + 1

### gaussian mask 생성 sigma = 10
sigma = 10
(X, Y) = np.meshgrid(np.linspace(0, N-1, N), np.linspace(0, M-1, M)) # 0~N-1(and M-1) 까지 1단위로 space를 만듬
Xc = np.ceil(N/2) # 올림 연산
```

```
Yc = np.ceil(M/2)
gaussianNumerator = (X - Xc)**2 + (Y - Yc)**2 # 가우시안 분자 생성

### low pass filter와 high pass filter 생성
LPF = np.exp(-gaussianNumerator / (2*sigma*sigma))
HPF = 1 - LPF

### LPF랑 HPF를 0이 가운데로 오도록iFFT함.
### 에너지를 각 귀퉁이로 모아 줌
LPF_shift = np.fft.ifftshift(LPF.copy())
HPF_shift = np.fft.ifftshift(HPF.copy())

### Log를 씌운 이미지를 FFT해서 LPF와 HPF를 곱해 LF성분과 HF성분을 나눔
img_FFT = np.fft.fft2(imgLog.copy(), (M, N))
img_LF = np.real(np.fft.ifft2(img_FFT.copy() * LPF_shift, (M, N))) # low frequency 성분
img_HF = np.real(np.fft.ifft2(img_FFT.copy() * HPF_shift, (M, N))) # high frequency 성분

### 각 LF, HF 성분에 scaling factor를 곱해주어 조명값과 반사값을 조절함
gamma1 = 0.3
gamma2 = 1.5
img_adjusting = gamma1*img_LF[0:rows, 0:cols] + gamma2*img_HF[0:rows, 0:cols]

### 조정된 데이터를 이제 exp 연산을 통해 이미지로 만들어줌
img_exp = np.expm1(img_adjusting) # exp(x) + 1
img_exp = (img_exp - np.min(img_exp)) / (np.max(img_exp) - np.min(img_exp)) # 0~1사이로 정규화
img_out = np.array(255*img_exp, dtype = 'uint8') # 255를 곱해서 intensity값을 만들어줌

### 마지막으로 YUV에서 Y space를 filtering된 이미지로 교체해주고 RGB space로 converting
img_YUV[:,:,0] = img_out
result = cv2.cvtColor(img_YUV, cv2.COLOR_YUV2BGR)
cv2.imshow('homomorphic', result)

cv2.waitKey(0)
```



더 연구해야 할 점:

1. 사실 목표는 화면 가운데의 천장에 반사되고 있는 빛이었는데 RGB값 확인 결과 차이가 많이 줄었다 하지만 여전히 꽤 차이가 났다.
2. 현재 방식의 filtering을 정확히 이해해야 할 것 같다. 그래야 다른 filtering을 적용하더라도 차이에 대한 이유를 알 수 있을 것 같다.
3. 이걸 이중주차 할 때 적용해 볼 수 있을 것 같다. 이중주차가 가능한 바닥면을 crop하고 바닥면과 다른 색을 가진 객체만 뽑아내고 그 객체를 객체추적하면 판단 가능할지도 모르겠다. 그 때 바닥면의 색이 비슷해야 차와 바

닥면을 정확히 구별할 수 있기 때문에 빛 반사는 계속해서 없애려고 노력해봐야겠다.

3. 연구 주제 : 서버(Linux)끼리 통신

선택 이유 :