|  |  |  |  |
| --- | --- | --- | --- |
| **베스텔라랩 R&D 연구보고서** | | | |
|  | | | |
| **연구제목** | yolo 및 MOT(moving object tracking) 최적화 | | |
| **연구목적** | 영상 처리 속도 개선과 계산량 감량 | | |
| **작성주차** | 1월 2주차 | **작성자명** | 서동환 현장실습 |
|  | | | |
| **에러노트** | tracking을 놓쳤을 때에 대한 예외 처리 미숙 | | |

### 문제 의식 :

차량의 주차면 in/out을 확인하기 위해 차량 tracking 구현하기로 했다. 인수인계서에 따르면 이미 MOT를 활용해 find\_intersection & trace\_inout 함수를 구현해 놨는데, 왜 사용을 안하는지에 대한 의구심이 들었다.

이유는 MOT의 계산량이 너무 많아서 200대가 넘는 CCTV를 multi-thread로 돌리면 영상 처리가 밀리기 때문이었다.

따라서 더 좋은 tracking 성능의 MOT보다는 더 빠른 MOT를 구현할 필요성을 느꼈다.

### - 해결 방안 :

계산이 빠르다는 것은 영상 처리가 빠르다 라는 뜻과 같기 때문에 더 빠른 MOT를 구현한다면 더 높은 수치의 FPS가 측정될 것으로 예상했다. 따라서 기존 사용하던 StrongSORT의 FPS를 파악하고, 다른 MOT 방식들과의 FPS 비교를 통해 가장 빠른 MOT를 파악하기로 했다.

사용한 MOT 종류&결과 (코드 생략):

yolov4 FastMOT & FPS 32

yolov5 StrongSORT & FPS 29

yolov5 Sort & FPS 32

yolov5 Towards-Reality-MOT & FPS 28

yolov5 DeepSort & FPS 11

yolov7 Byte-tracker & FPS 15

yolov7 DeepSort & FPS 11

우리가 보통 실시간 재생(Real-Time)이라고 느끼는 FPS가 30 이상이기에 FPS가 30이상 나오는 MOT를 선택하면 해결될 줄 알았는데, 기존에 계산량으로 문제를 일으킨 StrongSORT가 이미 좋은 FPS를 가지고 있었다. 즉 다른 MOT로 바꾼다고 한다고 해결될 문제가 아니라는 점이다.

다시 문제점을 찾기 위해 이번에는 yolov5로 그냥 object detection만을 진행했는데 그때의 FPS가 32가 나왔다. 이로서 MOT를 하느라 계산량이 많아진 것이 아닌 그냥 yolov5 자체가 계산량이 많아 object detection만으로 계산량이 포화인 상태인 것을 알게 되었다.

1. 문제 의식 :

MOT의 알고리즘을 고칠게 아닌 yolo의 속도를 고쳐야 한다.

- 해결 방안 :

* yolov5 가중치 개수 줄이기

딥러닝은 결국 행렬 계산만 진행한다. 따라서 모델 안의 가중치 개수가 줄면 자연스레 행렬 계산도 줄고, 결과적으로 계산 속도가 빨라진다. 다만 yolov5중 가장 가벼운 모델인 yolov5s를 사용했을 때의 결과가 위에서 나온 FPS32라는 점이다. 따라서 custom으로 yolov5s보다 가벼운 모델을 만들어야 한다는 것인데, 쉽지 않을 것 같다.

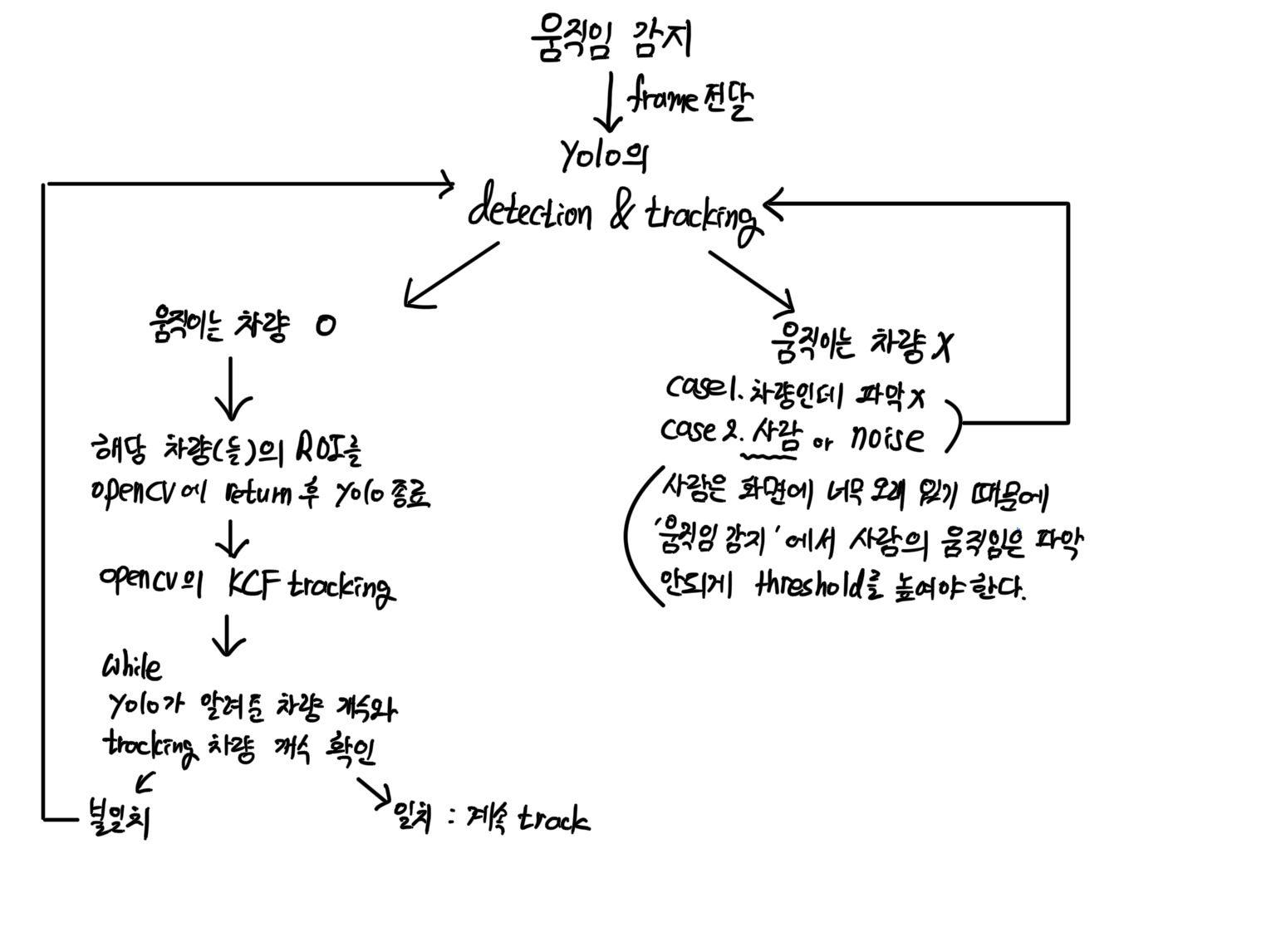
아마 custom을 만들게 된다면, model은 가볍게 dataset은 많게 학습을 시켜야 할 텐데, 그렇게 학습을 진행하게 되면 overfitting이 심하게 발생할 것이다. 다행인 점은, 주차장이라는 변화가 많지 않은 한정된 공간 안에서만 사용하게 될 모델이기 때문에 overfitting이 치명적으로 다가오진 않을 확률이 높다.

* yolov5 사용 줄이기

2-1. opencv 사용하기

위에서 MOT들의 결과를 봤을 때 object detection의 FPS 32, 거기에 MOT를 추가한것도 FPS 32라면 사실상 MOT는 시간을 쓰지 않는다는 것을 알 수 있다. 그렇다면 object detection으로 움직이는 차량을 검출하자마자 yolo를 놔버리고 tracking만 하면 되지 않을까 라는 생각을 하게 됬다. 문제는 MOT는 연속된 object detection의 결과가 없다면 사용할 수 없다. 이건 위의 MOT가 딥러닝에 맞춰진 tracker이기 때문에 그렇고 openCV를 사용하면 yolo가 움직이는 차량의 ROI값을 한번 만 보내주면 되기 때문에 위의 생각을 구현할 수 있을 것 같다.

-Code Flow



-구현 현황-

각 부분들은 거의 다 구현 되었는데, 이를 def 또는 Class로 합치는데 시간이 걸리고 있다. (코드 생략)

- 위 방식의 예상되는 문제점은

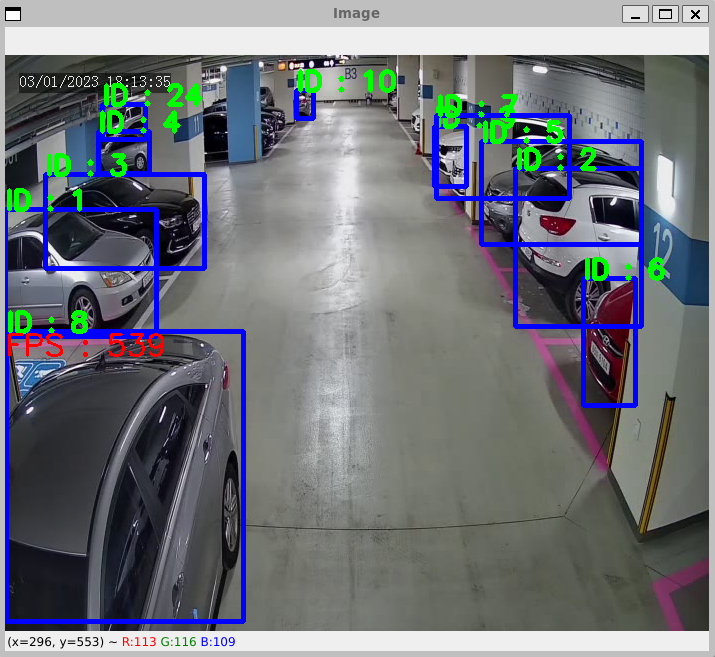
1. 일반적인 상황에 대한 처리는 가능할텐데 극한 상황에서 문제가 발생할 수도 있다.

2. 과도한 cpu사용. GPU사용을 최소화 하려고 openCV의 사용량을 늘렸는데, opencv는 gpu로 못 돌리기 때문에 오히려 GPU는 안 쓰이고 cpu만 과하게 쓰이는 현상이 생길 수도 있다.

3. KCF가 완벽한 tracking이 보장되지 않는다. 특히 차량이 보이는 면이 갑자기 바뀌면 대부분 tracking을 놓친다. 문제는 주차장에서는 대부분 세로로 들어와서 주차할 때 가로로 방향을 바꾸기 때문에 주차 직전에 놓치는 경우가 많다.일단은 좀 더 다양한 상황에 대한 영상으로 test를 해봐야 할 것 같다.

2-2. 일부 frame만 사용하기

차량을 detection & tracking하는 이유는 차량의 in/out을 판단하기 위함이지 사람이 영상을 보기 위함이 아니다. 따라서 사람 기준으로 영상이 연속적일 필요가 없다. 단지 yolo가 연속적으로 판단할 수 있을 정도의 frame만 넘겨줘도 된다. 그래서 이번에는 매 3개의 frame마다 yolo에게 넘겨주어 기존보다 yolo가 detection해야할 frame의 개수가 1/3이 되게하니 속도가 굉장히 빨라졌다.



기존에 yolo로 detection만 했을 때 FPS 14(yolov5m기준)가 나왔었는데, 위 방식대로 진행하게 되면 detection & tracking까지 하더라도 FPS 540까지 나온다.

이렇게 frame을 일부만 주게되면 화면에 엄청 큰 변화가 나타났을 때 해당 frame이 누락 되게 된다면 정보 손실이 클 것이다. 하지만 우리는 변화가 크지 않은 지하주차장만을 볼 것이기 때문에 크게 문제가 없을 것으로 예상된다.