복잡한 환경에서 영상 기반의 다중물체 추적기술 보완

DeepSORT에서 pedestrian의 object permanence 개선

Index

1. Background

1.1. 주제 선정

AI를 활용한 기술들이 발전하고 있지만 아직 미흡한 점들이 존재한다. 자율주행과 같이 보행자나 차량을 추적하는 상황에서 object의 이동을 예측할 때 다른 물체에 의해 occlusion되어 있으면 발생하는 문제가 대표적이다. 해당 object를 정확하게 tracking하지 못해 ID switching이나 이전에 탐지된 object를 새로운 object로 인식하는 등 발생하는 여러 문제를 확인하고 Multi-Object Tracking(MOT) 중 하나인 DeepSORT에서 해당 문제를 개선한다.

1.2. Why DeepSORT

Transformer 기반의 MOT가 SOTA에서 상위를 차지하고 있지만 많고 복잡한 연산이 필요하며 real time으로 적용 가능한 CNN 기반 MOT가 좋다고 판단하여 그 중 평가가 좋은 DeepSORT를 선택했다.

1.3. SORT (Simple Online and Realtime Tracking)

실시간 추적을 위해 Object들을 효율적으로 연관(Associate) 지어주는 MOT

1.3.1. Flow chart

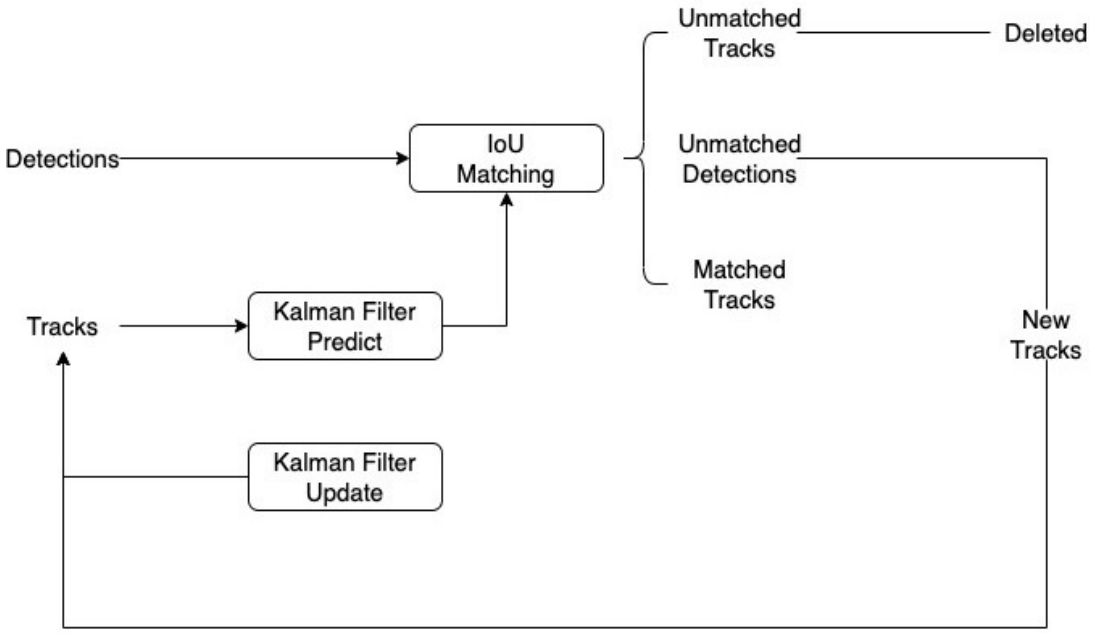


Figure 1. Flow chart of SORT

Detection: 프레임에서 객체를 탐지

Estimation: Kalman Filter를 통해 추적을 위한 측정치 예측, 업데이트 과정을 진행

Data Association:

i. IoU 유사도를 구함

ii. 추적되고 있던 객체와 추적되지 않는 객체(사라진 객체, 새로운 객체)를 분류

iii. 추적되고 있던 객체는 다시 Kalman Filter를 통해 다음 예측 값으로 사용

iv. 사라진 객체(Unmatched Tracks)는 일정 시간 이후 삭제되며, 새로운 객체(Unmatched Detections)는 새롭게 Track 생성 후 Track에 추가

1.3.2. SORT의 문제점

Kalman filter 효율성에도 불구하고 실제 상황에서 발생하는 occlusion, different view point, ID Switching에 취약하다.

2. DeepSORT

Deep Appearance Descriptor로 Re-identification 모델을 적용해서 ID Switching 문제를 해결하고 Matching Cascade 로직으로 더 정확한 추적을 가능하게 한다.

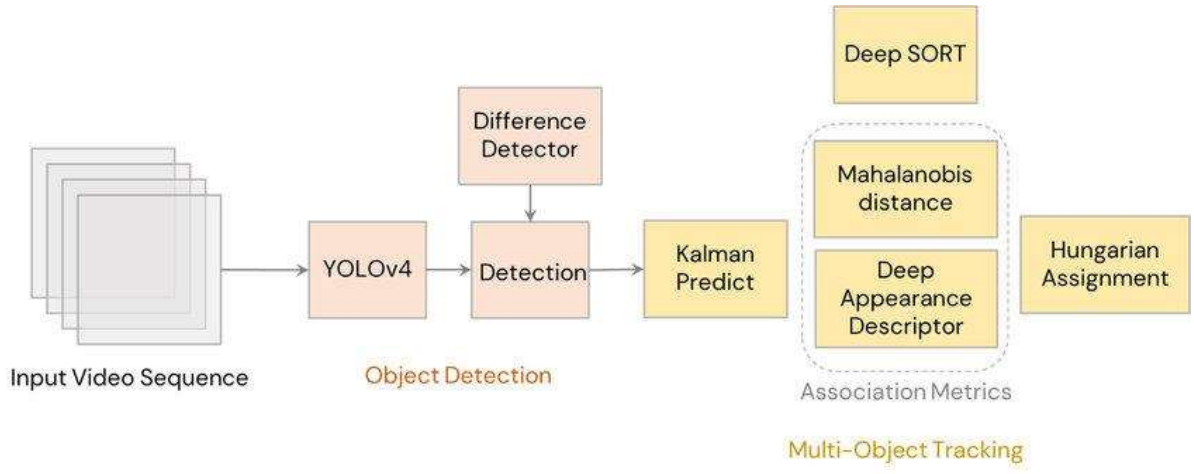


Figure 2. Flow Chart of DeepSORT

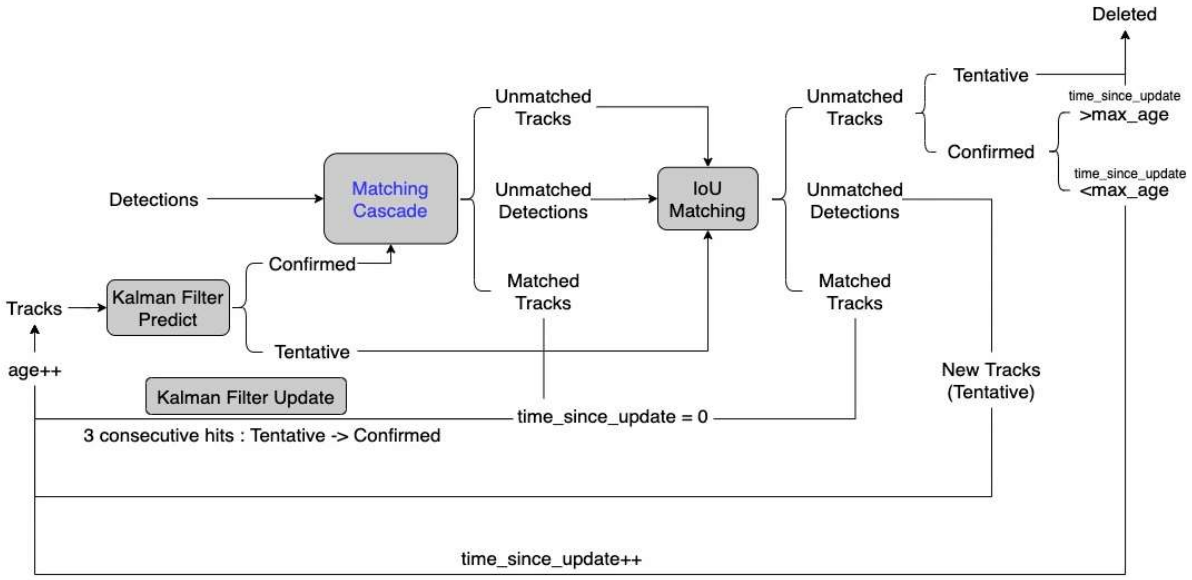


Figure 3. Flow Char of DeepSORT

2.1. Process of DeepSORT

1. Detection

- 이미지 input으로 받아 존재하는 물체에 대해 Bounding Box 정보를 받음

2. Kalman Filter Predict

- Kalman Filter를 통해 기존 Track 정보로부터 다음 frame 물체 위치를 예측

3. Track Check

- 해당 Tracks들이 충분한 근거 가진 Track인지 확인
- 3번 이상 물체 탐지해 Track으로 인정될 경우 "Confirm", 그렇지 않을 경우 "Tentative" 로 표현

4. Matching Cascade

- Confirmed인 Tracks에 대해 Matching 진행
- 이전보다 정확한 Track 정보 추출할 수 있음

5. IoU Matching

- Match되지 않은 Track, Detection들에 대해 기존의 SORT에서 사용한 IoU 매칭을 진행
- 새롭게 발견된 Detection에 대해 적용시켜주기 위함
- Matching하지 못한 Track들에 대해 보완하기 위한 목적도 있음

6. Tracking Life Cycle

- 매칭 결과를 기반으로 Track 객체의 생애주기를 정해줌

6-1. Unmatched Tracks (사라진 객체)

- 아예 잘못 판단한 경우는 삭제
- 기존에 Tracker로 잘 활동하는 녀석의 경우 일단 대기를 진행하다가 오랜 시간 살아 남아있을 경우(age) Tracker의 목적을 다한 것으로 판단하고 삭제
- 그렇지 않은 경우 물체가 다시 등장할 가능성이 있으므로 Tracks에 다시 추가
- 업데이트가 된 이후 탐지되지 못한 시간을 나타내는 변수 time_since_update를 1 증가

6-2. Unmatched Detections (새로운 객체)

- 새롭게 탐지된 객체이므로 Track을 생성
- 하지만 잘못 판단된 객체일 수 있으므로 상태는 "Tentative"로 설정

6-3. Matched Tracks

- 업데이트 된 이후 지속된 시간을 나타내는 변수 time_since_update를 초기화

7. Kalman Filter Update

- 현재 가진 Matched Tracks를 다음 frame 위해 Bounding box를 예측
- 이때, Track의 정당성을 주기 위해 Track이 등장한 횟수(hit)가 3회 이상 나왔을 경우 상태를 "Confirmed"로 변경

8. Recursive

- 이 과정을 재귀적으로 진행

2.1.1. Matching Cascade

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \ldots, N\}$, Detection indices $\mathcal{D} = \{1, \ldots, M\}$, Maximum age $A_{\max}$

Compute cost matrix $\boldsymbol{C} = [c_{i,j}]$ using Eq. 5
Compute gate matrix $\boldsymbol{B} = [b_{i,j}]$ using Eq. 6
Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
for $n \in \{1, \ldots, A_{\max}\}$ **do**
    Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
    $[x_{i,j}] \leftarrow \text{min\_cost\_matching}(\boldsymbol{C}, \mathcal{T}_n, \mathcal{U})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i,j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
end for
return $\mathcal{M}, \mathcal{U}$

1. Detections와 Tracks 사이의 cost matrix 계산

2. Detections와 Tracks 사이에서 cost를 기반으로 threshold를 적용하기 위한 mask 계산

3. 'Matched' matrix - 공집합으로 초기화

4. 'Unmatched detections' matrix - Detections으로 초기화 (매칭 시 제거)

5. 1부터 age까지의 값이 할당된 변수 n으로 for문

6. 마지막 n개 프레임에서 detection과 associate되지 않은(age=n인) track들을 부분집합으로 설정

7. 그 부분집합의 track과 매칭되지 않은 detection $\boldsymbol{u}$ 사이의 최소 cost 매칭

8. mask의 threshold 적용 후 match matrix에 넣기

9. 조건에 따라 unmatched matrix $\boldsymbol{u}$ 제외

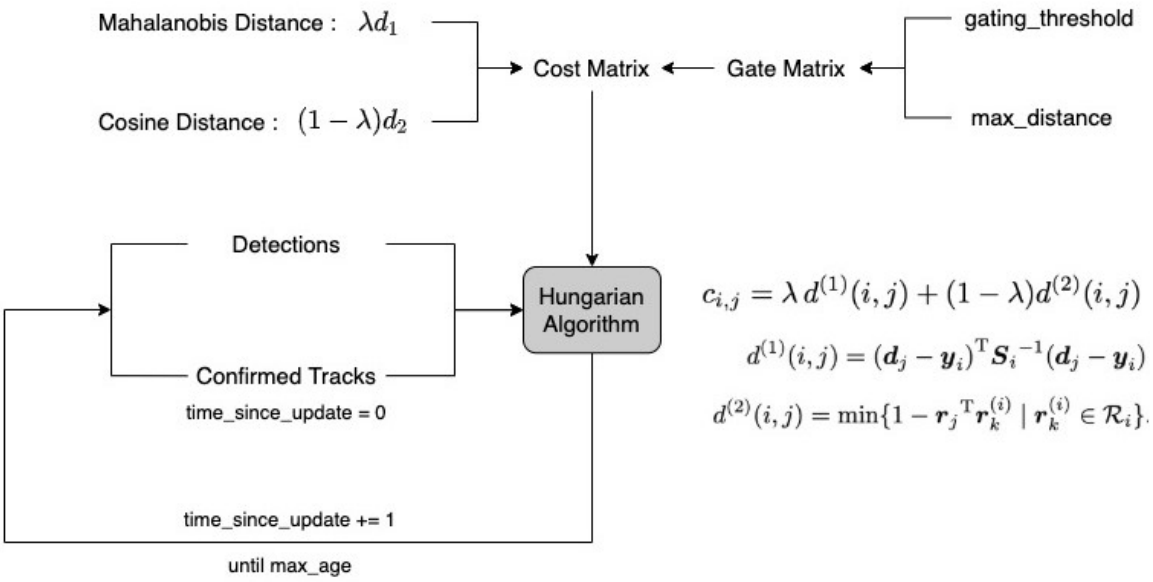


Figure 4. Flow Char of Hungarian Algorithm

Distance 1: Mahalanobis Distance

$$d^{(1)}(i,j) = (d_j - y_j)^T S_i^{-1} (d_j - y_j)$$

d_j : j번째 bounding box detection

y_j : track data의 평균값

S_i : track data의 공분산 행렬

Distance 2: Cosine Distance

물체가 가지는 모양을 판단하기 위한 Distance(Cosine 유사도를 활용한 distance)

$$d^{(2)}(i,j) = \{1 - r_j^T r_k^{(i)} \mid r_k^{(i)} \in \ R_i\}$$

1. 각 d_j에 대해 절대값이 1인 r_j(appearance descriptor: 외형 유사도를 판별하는 척도) 계산

2. 각 트랙 k에 대한 apperance descriptor 마지막 100개를 가지는 갤러리 r_k를 keep해둠

3. apperance space에서 i번째 track과 j번째 detection 사이의 제일 작은 cosine distance를 측정

예측된 track의 motion 동일한 정도가 높을 경우, 이를 반영하는 지표로써 사용하게 됨

장기간 occlusion 이후 동일성 회복에 유용

hyper parameter λ를 이용하여 가중치를 조절하여 사용

2.1.2. Deep Appearance Descriptor

CNN Architecture를 사용해 $d^{(2)}$에 들어가는 appearance 유사도를 측정하는 feature를 측정

$d^{(2)}$와 더불어 SORT 알고리즘에 Deep Learning이 필요한 당위성을 보여주는 가장 큰 부분

깊지 않은 CNN층을 추적하고자 하는 object 외형의 feature를 추출하는데 사용함으로써 real time을 가능하게 하며 이를 통해 외형 유사도 r_j는 0~1의 값을 갖게 되어 대상을 추적하는 값으로 사용

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Table 1: Overview of the CNN architecture. The final batch and ℓ_2 normalization projects features onto the unit hypersphere.

2.2. Quantitative Result of DeepSORT

		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	**71.0**	**80.2**	**46.9%**	21.9%	434	**587**	7880	**44564**	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	**406**	642	**5119**	63352	3 Hz
EAMTT [20]	**ONLINE**	52.5	78.8	19.0%	34.9%	910	**1321**	**4407**	81223	12 Hz
POI [16]*	**ONLINE**	**66.1**	79.5	**34.0%**	20.8%	805	3093	5061	**55914**	10 Hz
SORT [12]*	**ONLINE**	59.8	**79.6**	25.4%	22.7%	1423	1835	8698	63245	**60 Hz**
Deep SORT (Ours)*	**ONLINE**	61.4	79.1	32.8%	**18.2%**	**781**	2008	12852	56668	40 Hz

Table 2: Tracking results on the MOT16 [15] challenge. We compare to other published methods with non-standard detections. The full table of results can be found on the challenge website. Methods marked with * use detections provided by [16].

DeepSORT는 test에서 $\lambda = 0, A_{max} = 30$프레임으로 설정

Detection의 경우 confidence score에 대한 threshold는 0.3으로 설정

2.3. Pros and Cons

장점

매우 빠른 Object Detector 덕분에 빠르게 추적하며 real-time 애플리케이션에 사용 가능

높은 정확도를 보이며, SORT에 비해 ID Switching이 감소

단점

CPU와 GPU가 모두 필요하다. CPU는 계산이 느리고 실시간 처리에 적합하지 않고, GPU는 자체적으로 구동할 수 없어 cost가 크다.

feature에 배경 정보가 너무 많아서 Object Detector의 bounding box가 너무 크게 잡힐 때 알고리즘의 효과가 저하된다.

어두운 환경에서는 성능이 약간 저하된다.

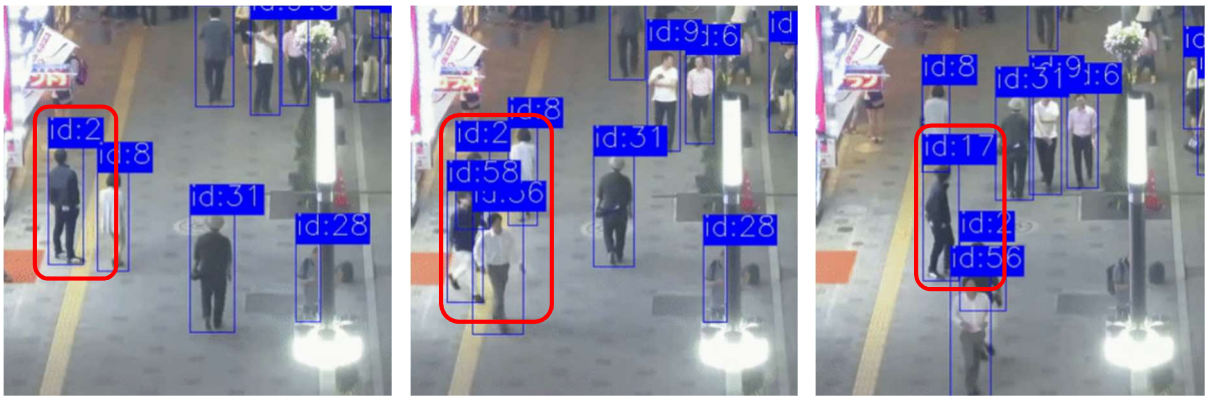
3. Evaluation of DeepSORT

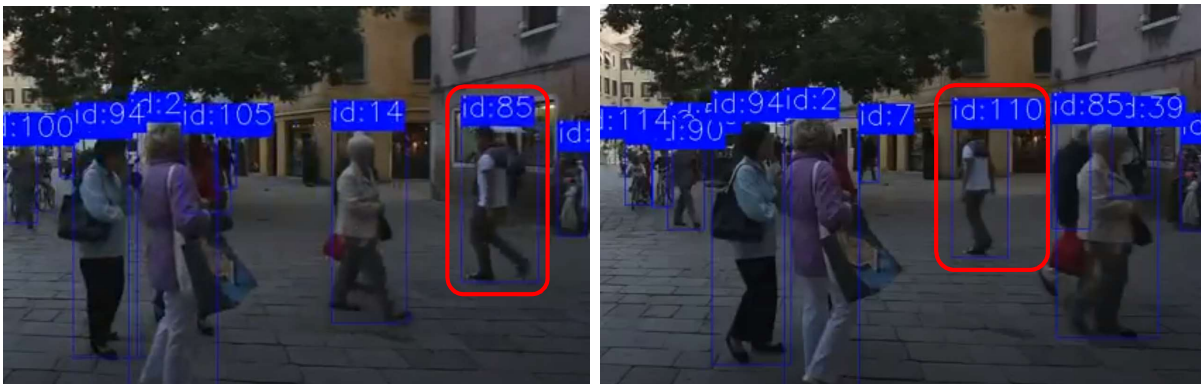
Dataset: MOT17

Detection model: YOLOv5

MOTA ↑	MOTP ↑	MT ↑	ML ↓	IDs ↓	FM ↓	FP ↓	FN ↓
32.3	79.7	28	94	293	782	5407	47743

4. Inference of DeepSORT





두명의 pedestrian들이 서로 엇갈려 지나가는 경우에 대해서는 SORT에 비해 ID switching이 감소했지만 세명 이상 서로 엇갈려서 만나 흩어지는 경우에 대해 ID switching과 새로운 object로 새로운 ID로 인식하는 것을 확인할 수 있다.

5. Analyze

5.1. 문제 분석

DeepSORT에서의 pedestrian의 object permanence 개선을 위한 방안을 고안했다.

4~5개 이상의 여러 object가 한꺼번에 교차에서 지나가는 경우를 완벽하게 모든 object에 대해서 tracking을 수행할 수 없지만 2~3개의 object에 대해서는 occlusion이 되어있는 unmatched track에 대해서 tracking을 할 수 있도록 알고리즘을 변경한다.

DeepSORT에서는 object가 겹치게 될 경우 detection이 되지 않으므로 match가 이루어지지 않아 age에 따라 tracking이 되지 않는 물체를 삭제하기 때문에 다시 인식 되었을 때 ID switching이 빈번하게 일어나게 된다. 따라서, 기존 DeepSORT 방식에서 가려진 물체를 계속 tracking 하기 위해 unmatched track 중 matched track과 계산했을 때, 설정한 IoU threshold 이상이고 IoT(Intersection of Track)가 가장 큰 track이 있으면 그 track에 가려져 있다고 판단하고 상태를 Occluded로 변경한다. 그리고 가려진 object도 가린 object와 위치가 겹치므로 가린 track과 match된 detection을 이용해 Occluded 상태로 된 track을 kalman filter를 사용해 update 해준다.

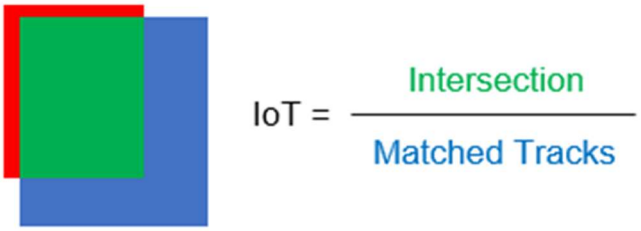


Figure 5. Intersection of Tracks (IoT)

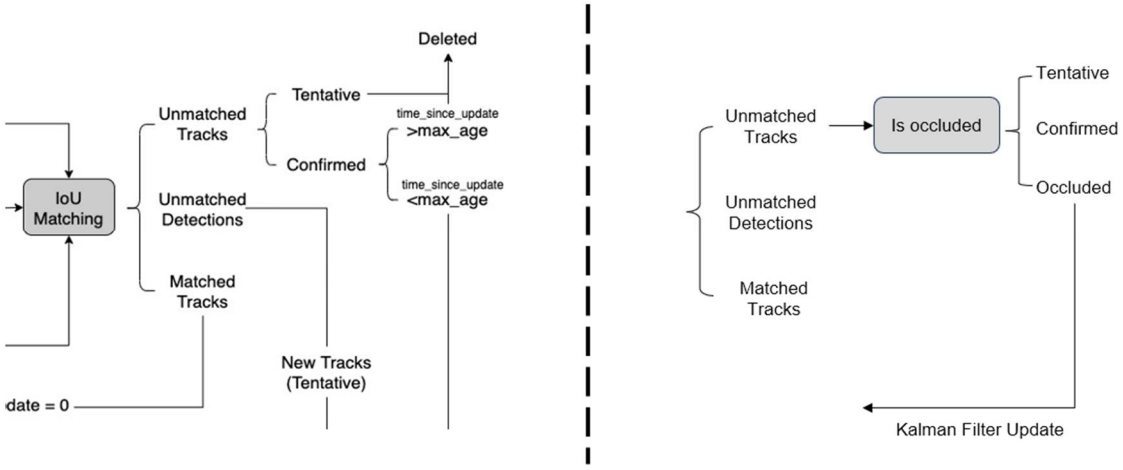


Figure 6. Flow Chart of Original DeepSORT and new methodology

5.2. 새로운 방법론

1. 기존 DeepSORT에서 IoU matching 이후 unmatched track에 대해 matched track과 유사도를 계산

2. unmatched track 중 matched track과 겹치게 되어 occlusion이 일어났다고 판단되면 unmatched track의 TrackState를 Occluded로 변경

3. Occluded 상태로 전환된 track을 겹친 matched track과 match 된 detection을 이용해 kalman filter update

6. Result

6.1. Quantitative Result

Dataset: MOT17

Hyperparameters

- max_age = 60
- IoU_threshold = 0.5
- IoT_threshold = 0.7 (Intersection of Track)

	MOTA ↑	MOTP ↑	MT ↑	ML ↓	IDs ↓	FM ↓	FP ↓	FN ↓
Original	32.3	79.7	28	94	293	782	5407	47743
New	32.5	79.8	27	96	258	703	4886	48167

ID switching이 소폭 감소했으나 기존의 DeepSORT의 ID switching을 크게 보완했다고 하기엔 조금 부족한 수치이다.

6.2. Qualitative Result

4. Inference of DeepSORT에서 result와 다르게 id: 2의 보행자가 id switching이 발생하지 않는 것을 확인할 수 있으며 id: 6의 보행자도 id: 2의 보행자와 겹친 후 그대로 id를 유지하는 것을 확인할 수 있었다.

7. Conclusion

Pedestrian에 대해서 object permanence를 유지해 DeepSORT의 ID switching을 감소하고자 했고 새로운 방법론을 통해 특정 parameter에서 소폭 감소하는 경향을 보였다. 하지만 크게 감소시키지 못했다.

새로운 방법론으로는 object permanence를 완전하게 부족한 부분이 있다. Object가 occlude된 상태에서 선형적으로 움직이며 해당 object를 가린 matched track과 같이 움직인다고 가정했기에 가정이 모순인 경우가 발생할 수 있다. 또한 2~3개의 object에 대해서는 object permanence를 유지하게 한다 하더라도 그 이상의 object들이 겹치는 문제는 해결할 수 없는 한계도 존재한다.

8. Reference

https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss11/OPT/lec11.pdf

https://kr.mathworks.com/help/stats/mahal.html

Simple Online and Realtime Tracking (https://arxiv.org/pdf/1602.00763v2.pdf)

Simple Online and Realtime Tracking with a Deep Association Metric

(https://arxiv.org/pdf/1703.07402v1.pdf, https://github.com/nwojke/deep_sort)

Measurement-wise Occlusion in Multi-object Tracking (https://arxiv.org/pdf/1805.08324.pdf)

MOT16: A Benchmark for Multi-Object Tracking (https://arxiv.org/pdf/1603.00831.pdf)

YOLOv5 source code (https://github.com/ultralytics/yolov5)

DeepSORT source code (https://github.com/nwojke/deep_sort)

MOT16/17 Evaluation Toolkit source code (https://github.com/shenh10/mot_evaluation)