SDC Final Competition Report

Team Members -

學號 : A111137

姓名 : 邱柏鎧

Reference

1. Score refinement for confidence-based 3D multi-object tracking
2. CenterTrack
3. CenterPointDetection
4. Learning score update functions for confidence-based MOT

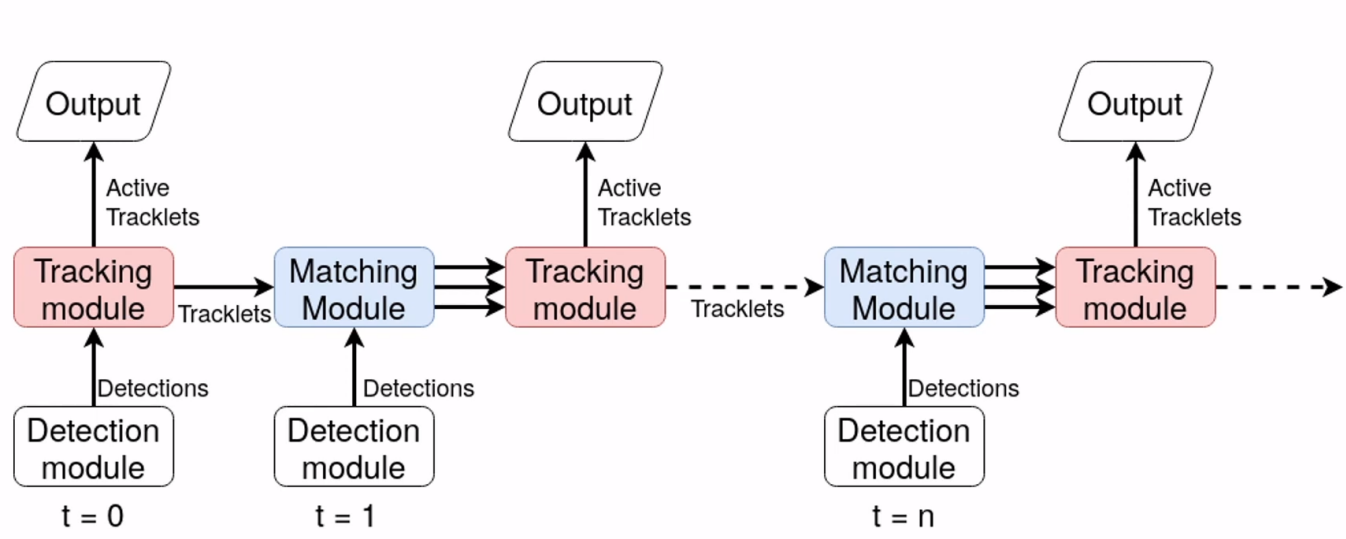
Contribution

先以結論而言，其實最主要影響成績的還是參數，所以我大部分的時間也都是在測試這些參數，當然像是速度error的設計，是可以先用簡單的嘗試去調整，但實際精準的數字仍然需要大量時間測試 😄。以下會忽略調參的過程與測試，主要討論在程式以及架構上面的理解、改動，以及我為何會想使用這些方法。

Main pipeline of tracking case

在追蹤物件的情境之下，最主要可以切分成幾個模組，而整個流程圖如下所示。

- Detection module : 辨識物件在畫面中的位置，通常以bounding box框列，並且進行分類。
- Matching module : 將辨識的結果與上一時刻追蹤的物件進行比對。
- Tracking module : 根據比對的結果，決定哪些物件有成功追蹤，而失敗的需要保留或是淘汰。



此次的方法都建立在相似的架構當中，而tracking module細部又包含Tracklet buffer, Filtering algorithm, Tracklet management system. 在Filter的部份，經過測試發現KF成效不彰，所以最終維持使用Point tracker based的方法，純粹計算前後兩個frame中，detection物件的中心點距離。此架構最重要的部份是update function (tracklet management system中)，透過此函數的設計，判斷是否保留物件或是追蹤物體消失等狀況。而Update function可以粗略的劃分為兩類：

- **Count-based :** 根據成功matching的次數來決定啟用、保留或是丟棄此物件ID。最主要是min-hits和max-age等參數來當作threshold的數值。
- **Confidence-based :** 對每個物件都有設置信心值 (confidence)，配對前會扣予預先設置的數值 (score-decay)，而配對成功會使信心值增加 (增加數值由detection score決定)，反之則不會有變動，決定保留或淘汰的依據，同樣是用預先設定的參數 (threshold)。

Matching module

Baseline的Matching module是基於greedy search的演算法，其表現也超越許多NN based的方法。Greedy search又可以細部劃分為global與local的方式。經由測試後發現，global的方法在此次的dataset中表現較差(差異不大)，所以接下來的研究和改動會建立在**Matching module為local greedy search**。

```
def local_greedy(dist): # CBMOT
    matched_indices = []
    if dist.shape[1] == 0:
        return np.array(matched_indices, np.int32).reshape(-1, 2)
    for i in range(dist.shape[0]):
        j = dist[i].argmin()
        if dist[i][j] < 1e16:
            dist[:, j] = 1e18
            matched_indices.append([i, j])
    return np.array(matched_indices, np.int32).reshape(-1, 2)
```

Tracking module

原本Baseline使用的方法是count-based method。然而此方法面對物件只是被遮擋一下，或是當下detection不準確等因素，都有可能直接的捨棄掉原本追蹤的物件ID，就算是簡單的調整threshold參數也很難達到更好的成果。為了讓我們判斷的依據能更有可信度，改而採用**confidence-based**的方法。

```
## To reach better result, you should
## modify these parameter
NUSCENE_DECAY_NOISE = {
    'car': 0.06,
    'truck': 0.1,
    'bus': 0.06,
    'trailer': 0.075,
    'pedestrian': 0.175,
    'motorcycle': 0.05,
    'bicycle': 0.1,
    'construction_vehicle': 0.075,
    'barrier': 0.075,
    'traffic_cone': 0.075,
```

```
}
...
for m in matched:
    ...
    self.noise = NUSCENE_DECAY_NOISE[self.tracks[m[1]]['detection_name']]
    self.tracks[m[1]]['detection_score'] = np.clip(self.tracks[m[1]]
['detection_score'] - self.noise, a_min=0.0, a_max=1.0)
    # update detection score
    track['detection_score'] = update_function(self, track, m,
self.loaded_model)['detection_score']
    ret.append(track)
...
for i in unmatched_trk:
    track = self.tracks[i]
    # update score (only apply score decay)
    if self.score_update is not None:
        self.noise = NUSCENE_DECAY_NOISE[track['detection_name']]
        if track['detection_name']=='bicycle':
            self.noise = 0.075
        track['detection_score'] -= self.noise
```

每個tracklet在frame配對前，都會被扣予一個數值 (noise, NUSCENE_DECAY_NOISE)，配對失敗的次數太多的話，信心值下降，超過預先設定的threshold就被丟棄。反之如果配對成功，則會增加其信心值，而增加的依據來自於detection_score。至於細部的更新方法，就是藉由設計update_function來決定，最終有兩個方法表現較為出色，以下分別介紹:

1. **multiplication**

$$s_t^{Tk} = 1 - ((1 - s_{t-1}^{Tk}) \cdot (1 - s_t^{Dt}))$$

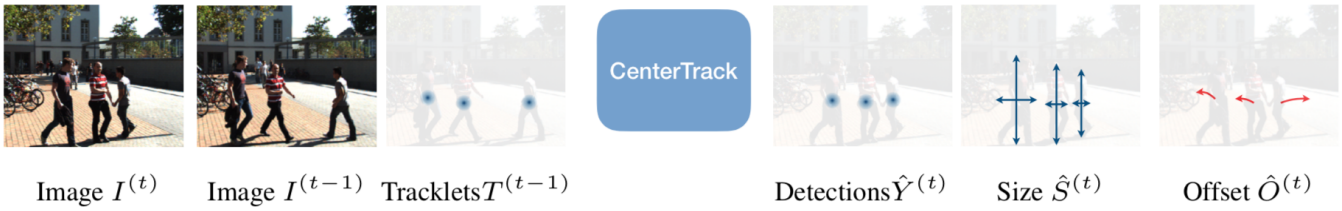
   根據論文所述，update_function最核心的觀念是要滿足"當match成功，則confidence要增加"的條件，所以可行的方法就有很多種，其中表現最好的是上面所列出的公式。由公式可知，每次update的時候會根據前一個frame的tracking score和當下的detection score，來決定當下的tracking score。換個角度去解釋，因為 s 代表著**Detection**或是**Tracking**的信心程度，所以$(1-s)$就代表著**uncertainty**，也就是，t 時刻Tracking的uncertainty就會是 t-1 時刻Tracking的uncertainty乘上 t 時刻Detection的uncertainty。

2. **Neural Network based**

   原本的update_function建立在我們對於更新的趨勢有一定的猜想，然而實際上背後的更新模型沒有我們想像的簡單。於是套用最常見的MLP當中網路架構，套用了4層hidden layer，藉由數據的訓練，得到一個相對合適的weight和bias。而最後通過調整decay的參數以及NUSCENE_CLS_VELOCITY_ERROR的修正，就可以達到非常好的成績。

Camera based tracking - CenterTrack

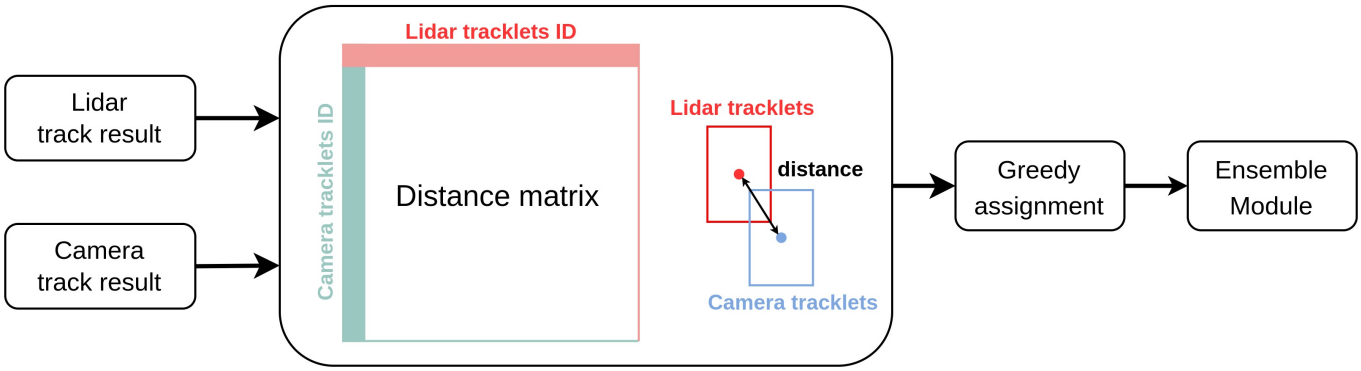
前面所提到的方法，建立在detection是使用lidar的資料，並且透過center point detection來實踐。儘管調整參數就可以讓最終的成果達到 **AMOTA=0.707** 的成績，但如果想要更進一步提升成績，勢必需要更多的改動，而最簡單的方法就是讓感測的資料不再只有Lidar，於是透過論文的搜尋，我們可以找到基於Camera based的一些方法，此次使用的是Center Track的技巧。



Center track的輸入為 t 時間的圖像,以及 t-1 時間的圖像和heatmap(物體中心分佈的熱力圖)，網路架構簡單來說會有上下採樣的過程提取特徵，最終得到四個feature map，分別為 t 時間的heatmap、confidence map、size map以及offset map。此網路最大的優勢就是合併了檢測和匹配的過程，使其面對MOT的case能夠加速運算，不像很多論文處理的方法是detection用一個網路架構，tracking又用其他的架構。而這樣的優勢，自然也有trade-off，所以它犧牲了準確度，也因此在我的使用當中，會再透過greedy search，重新根據detection和tracking的結果來修正追蹤的id。

Fusion Method

透過camera和lidar的兩種不同感知方法，我們可以應付更多的場景，有種截長補短的效果，但是如何融合這兩個tracking的成果也是值得討論的地方。傳統的ensemble方法就是使用多數決 (vote)，然而我們並沒有多數的狀況，因為只有camera和lidar參與投票。為此，改用late-fusion method來融合兩個tracking成果，其架構與方法很類似於原本lidar的tracking和matching module。



其中Ensemble的部份同樣考慮進去unmatch以及match的情況，不過根據測試，如果match的情況發生，則不用像前面lidar-based方法需要進行decay，唯獨在兩個tracklets無法對齊時，才會進行decay。而update_function的部份，則是修改原本所設計的公式為：

$$s_t^{Tk} = 1 - ((1 - s_t^{lidar}) \cdot (1 - s_t^{camera}))$$

Conclusion

其實這次的競賽，最主要是需要調整適當的參數，就可以達到預期的成果，如果參數設置錯誤，反而會讓AMOTA下降。而實際程式上的改動不用太多，就可以超過baseline。整個調整過程當中，NUSCENE_CLS_VELOCITY_ERROR 會是最主要更動的對象，基本上此數值決定了當距離超過多少（或說速度）要將其排除，而根據不同種類的交通工具、行人，就會有不一樣的參數，大致上可以藉由速度快慢的程度來進行調整。

另外，因為我使用了confidence-based的方法，所以對於不同種類的decay (noise)參數也應該調整，適當的設置之後，數值也會有所提高。Update_function的設置，如果使用multiplication，則在參數維持原本的狀況下，也可以到達AMOTA=0.688的表現，不過改用NN這種網路架構的方式後，反而讓調整參數的效果非常明顯，甚至可以超過0.7，而我使用此方法最好的成績為**AMOTA=0.707**。

就如同前面所提到，為了讓分數更進一步提升，最簡單暴力的方法就是引入其他感測器。儘管camera based的方法普遍無法達到較高的AMOTA（甚至0.2已經是很好的表現），在一些遮擋、形狀相似等等的case當中，視覺仍然可以彌補不足的部份。而在使用fusion的方法後，最終的AMOTA可以達到**0.724**的成績。