Final Competition: Tracking Report

1. Team members

311605007 簡紹恆
311512015 謝元碩

2. Contribution

(1) Greedy algorithm 設計理念

在這個演算法中，最主要的目的是要透過物件與物件之間的距離資訊，去尋找每一個追蹤的物件所配對到的物件 index。因此，如何篩選最有可能正確的 index 是此部分最大的挑戰。以下我們將針對幾個不同的算法做分析：

```
temp = np.argmin(dist, axis=1)
print(temp)
for i in range(len(temp)):
    matched_indices.append([i, temp[i]])
matched_indices = np.array(matched_indices)
print("matched_indices: ", matched_indices)

# # raise NotImplementedError("Greedy algorithm not implemented yet!")
# ### Student implement ###
# return matched_indices
return np.array(matched_indices, np.int32).reshape(-1, 2)
```

- 首先，我們嘗試取出每一行的距離最小值，並將最小值索引存入 matched_indices 中。這個算法的測試結果遠低於 baseline，因為我們將 matched_indices 輸出到 terminal 檢查後，發現會有非常多重複對應到的 index，多對一追蹤的狀況才導致分數很低。此部分的 AMOTA 僅有 0.616。

```
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])
print("matched_indices: ", np.array(matched_indices, np.int32).reshape(-1, 2))
return np.array(matched_indices, np.int32).reshape(-1, 2)
```

- 一開始只用助教留下來的 Hungarian Algorithm 會非常慢，電腦會因為記憶體不足而當機，而且出來的結果也不盡理想，所以後來使用 github 上找到的貪心算法，嘗試使他記憶體使用率小一點。
- 接下來，我們則是在配對過程中，不斷將「已經配對過」的

index 濾除(變成 1e18)，如此每一組配對才不會重複，當然過程遇到 1e18 的部分皆不會考慮到配對中。這個方法的結果，則通過 baseline，AMOTA 分數為 0.680，可知以 baseline 而言，僅需簡易的依序配對即可通過。

```
if dist.shape[1] == 0:
    return np.array(matched_indices, np.int32).reshape(-1, 2)
for i in range(dist.shape[0]):
    temp = np.argmin(dist, axis=1)
    min_val = 1e19
    for j in range(dist.shape[0]):
        if dist[j][temp[j]] < min_val:
            min_val = dist[j][temp[j]]
            min_dist = [j, temp[j]]
    if min_val < 1e16:
        dist[:, min_dist[1]] = 1e18
        dist[min_dist[0], :] = 1e18
        matched_indices.append(min_dist)

matched_indices.sort(key=lambda x:x[0])
print("matched_indices: ", np.array(matched_indices, np.int32).reshape(-1, 2))
return np.array(matched_indices, np.int32).reshape(-1, 2)
```

- 為了讓配對效果更加理想，我們繼續針對前述演算法做改良。由於前面是從第 1 個物件到第 n 個物件依序配對和濾除，我們認為這樣的決策方式有點粗糙，較早配對的物件，所配對到的距離最小值可能也並非理想，若其他物件正好與此物件擦身而過，極有可能造成配對到經過的他種物件，造成錯誤。
- 因此，我們在每一次的判斷中，再去尋找這些最小值距離的「最小值」，讓距離相對最近的物件先被配對到、加入 matched_indices，之後再繼續尋找每個行的最小值，再做同樣的判斷，以此類推，越後面配對到的物件，代表其配對到的最小值距離相對較大(有可能是其他物件路過導致)，我們認為這樣的算法是更加理想的。
- 然而，滿令人意外的是，經測試後發現分數低於 baseline，AMOTA 為 0.668。思考其原因，有可能是此配對方法擁有一些邏輯上的疏忽，配對距離較小的物件，似乎也有可能是他種物件路過所產生的距離，導致配對會越錯越多。我們認為應該再多考慮距離第二、第三小的部分，觀察這些數值與最小值的變化，以判斷是否有發生追蹤物件擦身而過的情況。此部分因時間因素，未能實現程式碼及做測試。

- 最後我們將前述所設計的貪婪演算法與匈牙利算法做比較。匈牙利算法較不同於貪婪演算法，是透過線性分析進行物件配對，不過由於線性分析所考慮因素少，匈牙利算法分數遠低於自己設計之演算法及 baseline。

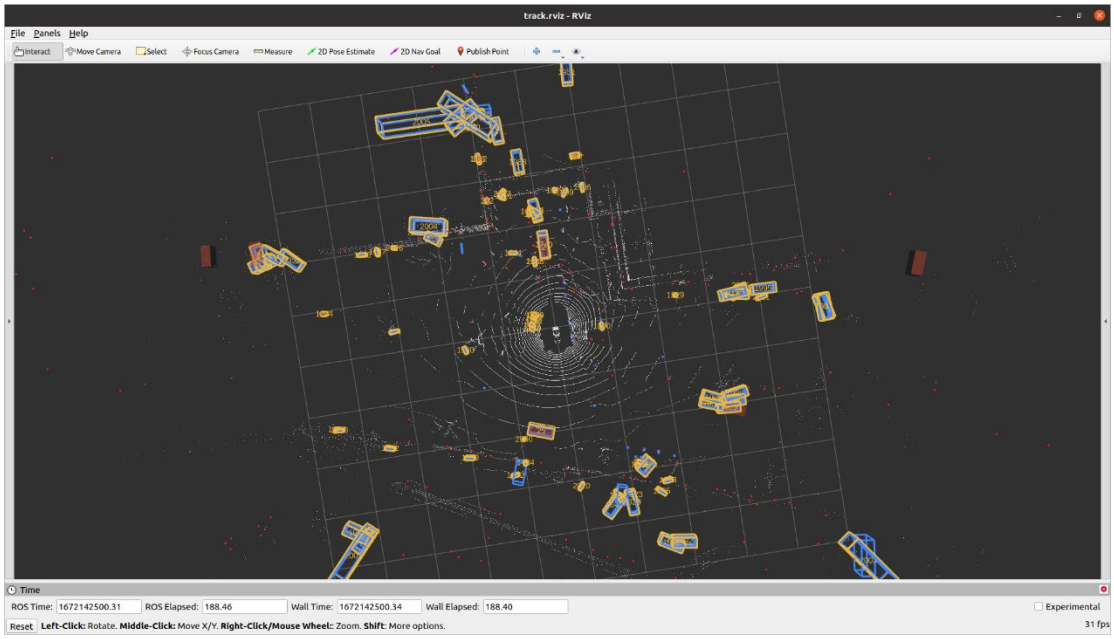
(2) 進行貪婪演算法前的距離數值濾除問題

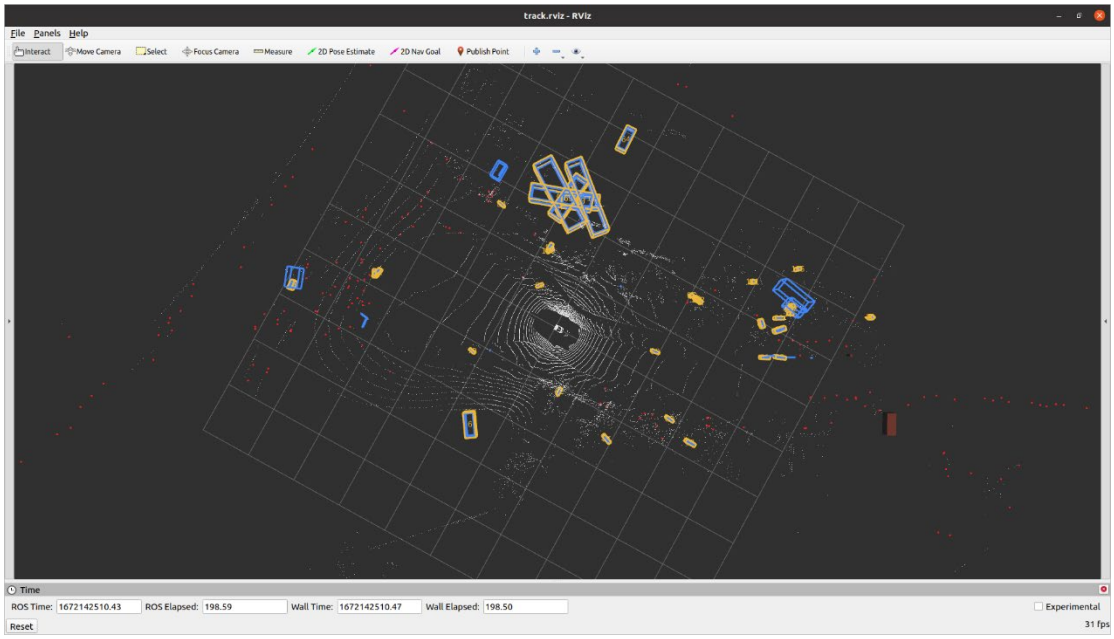
```
invalid = ((dist > max_diff.reshape(N, 1)) + (
        positions2_cat.reshape(N, 1) != positions1_cat.reshape(1, M))) > 0
dist = dist + invalid * 1e18
```

進行貪婪演算法前，程式會針對兩種狀況首先進行排除：當距離大於閥值，以及當 resource 和 track 物件符合時(避免用錯誤的距離作配對)，不符合狀況即會將數值全部加上 1e18 濾除掉。

```
NUSCENE_CLS_VELOCITY_ERROR = {
    'car': 3,
    'truck': 4,
    'bus': 5.5,
    'trailer': 2,
    'pedestrian': 1,
    'motorcycle': 4,
    'bicycle': 2.5,
    'construction_vehicle': 1,
    'barrier': 1,
    'traffic_cone': 1,
}
```

- Max_diff 閥值則來自此 dictionary，每樣物件都有其不同的通過閥值，而這會影響之後進入貪婪演算法的 dist 資訊，配對結果正確與否同樣受此閥值影響，因此我們嘗試修改此部分數值來增高追蹤準確度。由於這裡主要為調參，我們僅稍微描述調整過程及測試結果。
- 首先，測試後發現，降低 car 及 bicycle 能夠較明顯提升分數，為 0.681；另外，我們觀察到 trailer 的 MOTA 分數落在 0.44 左右，為主要拉低分數的追蹤物件，若能改善分數即可提升許多，然而分別提高或降低誤差數值後，未能改變其追蹤分數，初步認為可能原本的環境對於這兩樣物件本來就不好追蹤，或是被遮擋、樣本數少所導致。
- 剩餘的其他物件，經調整後分數未能有所提升，有些甚至有分數下降情況。因此後續的追蹤測試我們將 car 及 bicycle 降低 0.5 來執行，讓這兩項物件的定位更為精準，不會因為過於相近而吃到附近其他的同物件，此方法經測試後結果為 AMOTA=0.682(目前最佳解)。下圖是我們較佳追蹤結果的可視化界面：





3. Problems and Solutions

(1) 改用 KF 進行 center 預測

PointTracker 使用單位時間，以當下速度決定下一楨的位置，而這樣為線性預測，有可能造成預測成效不佳。程式中包含 KF(初始未開啟)，卡爾曼濾波器為一種以高斯分布為前提的非線性預測，甚至考慮觀察值，預測結果有機會提升，因此這裡嘗試使用 KF 對物件做追蹤預測。在 discussion 我們會針對此部分做討論。

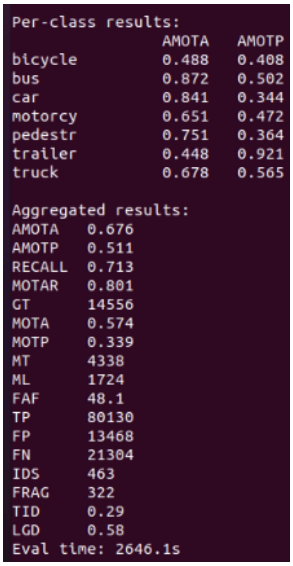
(2) 對 matched_indices 做排序的問題

當我們在做貪婪演算法的時候，在新增配對項時，有可能不會照順序新增，因此 Tracking_id、age、active 的排序也不會依照 index 原本順序。這可能影響後面 frame 的配對結果或是消失物件的 age 累加功能、active 數值。

針對這項問題，我們在進行自己設計過的貪婪演算法後，會使用 sort 指令將其依序第一列進行排序。

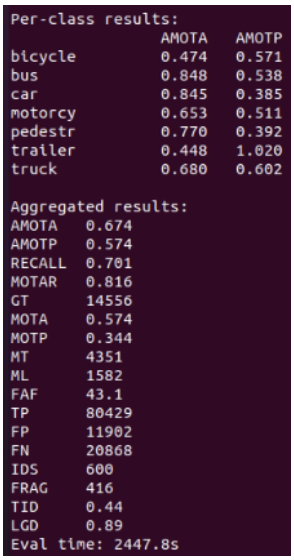
(3) Max age 問題

Max age 代表當一個物件原本在前一個 frame 中，但下一個 frame 消失時，max age 會記錄此消失物件 id 的存活時間，在期間內的物件仍然會透過速度持續追蹤及更新。我們認為此部分會間接影響到 frame 中所追蹤的 resource，其 dist 也會不同，所進行貪婪演算法後的結果也會不同，因此想嘗試更改這個參數。



```
Per-class results:
                AMOTA   AMOTP
bicycle         0.488   0.408
bus             0.872   0.502
car             0.841   0.344
motorcy         0.651   0.472
pedestr         0.751   0.364
trailer         0.448   0.921
truck           0.678   0.565

Aggregated results:
AMOTA   0.676
AMOTP   0.511
RECALL  0.713
MOTAR   0.801
GT      14556
MOTA    0.574
MOTP    0.339
MT      4338
ML      1724
FAF     48.1
TP      80130
FP      13468
FN      21304
IDS     463
FRAG    322
TID     0.29
LGD     0.58
Eval time: 2646.1s
```



```
Per-class results:
                AMOTA   AMOTP
bicycle         0.474   0.571
bus             0.848   0.538
car             0.845   0.385
motorcy         0.653   0.511
pedestr         0.770   0.392
trailer         0.448   1.020
truck           0.680   0.602

Aggregated results:
AMOTA   0.674
AMOTP   0.574
RECALL  0.701
MOTAR   0.816
GT      14556
MOTA    0.574
MOTP    0.344
MT      4351
ML      1582
FAF     43.1
TP      80429
FP      11902
FN      20868
IDS     600
FRAG    416
TID     0.44
LGD     0.89
Eval time: 2447.8s
```

上面左圖為降低 age 的結果，右圖則為提高 age 的結果。可以發現分數都沒有原本 age=6 時高，我們並沒有仔細推敲可能的原因，但初步猜測，此數值可能已經是經測試後的最佳解。後來我們仍使用 max age=6 來進行追蹤測試。

4. Work division

謝元碩：greedy algorithm、研究 KF、max_age 調整、報告
簡紹恆：可視化(因元碩電腦記憶體不夠)、調整 velocity error 參數、研究 KF、報告

5. Other Discussion or Findings

(1) KF 與 PointTracker 比較

Visualize 不管怎麼調參數都很難看出來到底有沒有改善，有些場景就是一定會產生多個物體重疊。因此使用預設的 KF 參數並不會讓結果變好，但是由於 Visualize 的畫面過於相似，調整無從下手。

```
Aggregated results:
AMOTA   0.681
AMOTP   0.530
RECALL  0.697
MOTAR   0.821
GT      14556
MOTA    0.576
MOTP    0.334
MT      4426
ML      1612
FAF     45.4
TP      80752
FP      13113
FN      20669
IDS     476
FRAG    339
TID     0.35
LGD     0.71
Eval time: 2782.4s
```

使用 pointtracker

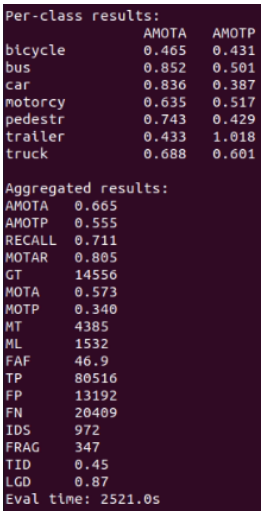
```
Aggregated results:
AMOTA   0.664
AMOTP   0.561
RECALL  0.715
MOTAR   0.797
GT      14556
MOTA    0.573
MOTP    0.342
MT      4387
ML      1522
FAF     47.5
TP      80580
FP      13265
FN      20345
IDS     972
FRAG    346
TID     0.45
LGD     0.91
Eval time: 2679.1s
```

使用 KF

```
Aggregated results:
AMOTA   0.682
AMOTP   0.530
RECALL  0.701
MOTAR   0.818
GT      14556
MOTA    0.578
MOTP    0.335
MT      4432
ML      1604
FAF     45.7
TP      80805
FP      13145
FN      20615
IDS     477
FRAG    340
TID     0.35
LGD     0.71
Eval time: 2769.9s
```

使用 pointtracker 並且將 car 跟 bicycle 的 velocity error 都降低 0.5

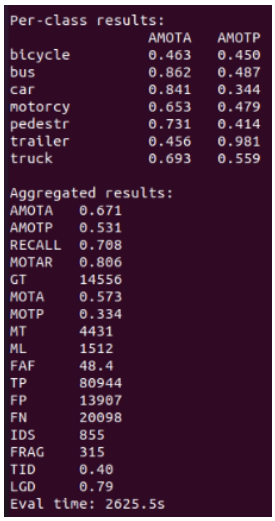
以下包含兩個部分：只對座標值做預測，以及對座標值及速度值做預測。



```
Per-class results:
                AMOTA   AMOTP
bicycle         0.465   0.431
bus             0.852   0.501
car             0.836   0.387
motorcy         0.635   0.517
pedestr         0.743   0.429
trailer         0.433   1.018
truck           0.688   0.601

Aggregated results:
AMOTA   0.665
AMOTP   0.555
RECALL  0.711
MOTAR   0.805
GT      14556
MOTA    0.573
MOTP    0.340
MT      4385
ML      1532
FAF     46.9
TP      80516
FP      13192
FN      20409
IDS     972
FRAG    347
TID     0.45
LGD     0.87
Eval time: 2521.0s
```

只有座標



```
Per-class results:
                AMOTA   AMOTP
bicycle         0.463   0.450
bus             0.862   0.487
car             0.841   0.344
motorcy         0.653   0.479
pedestr         0.731   0.414
trailer         0.456   0.981
truck           0.693   0.559

Aggregated results:
AMOTA   0.671
AMOTP   0.531
RECALL  0.708
MOTAR   0.806
GT      14556
MOTA    0.573
MOTP    0.334
MT      4431
ML      1512
FAF     48.4
TP      80944
FP      13907
FN      20098
IDS     855
FRAG    315
TID     0.40
LGD     0.79
Eval time: 2625.5s
```

考慮座標跟速度

- 加入速度作為預測因素後，AMOTA 有較高一點，然而兩者都未過 baseline，可以推測我們沒有調整濾波器參數導致，包括雜訊及係數。由此結果可以得知，多考慮一個輸入可使追蹤結果更加精確。
- 另外在調整 KF 時，我們觀察可視化界面發現車子容易卡在一起，所以我們有針對卡在一起狀況較多的物件，調整其 velocity error，嘗試讓有考慮速度的 KF 效果變好。然而只要是單純開啟 KF 功能來使用，以及調整 velocity error，AMOTA 分數仍然很低，由於時間不足的關係，此部分是個很值得去探討的議題。