

SDC Tracking Competition

A. Pipeline

來源: https://github.com/apak-00/argoverse_simple_gmf_tracker

1. Detection:

直接使用 `argoverse_detections_2020` 提供的結果, 內容包含: 被 label 物件的 pose, 3D boundingbox 的長寬高, 時間戳記等

```
{
  "center": {
    "x": 10.893529891967773,
    "y": -2.9378652572631836,
    "z": 0.44712644815444946
  },
  "height": 1.4641387462615967,
  "label_class": "VEHICLE",
  "length": 4.45181941986084,
  "occlusion": 0,
  "rotation": {
    "w": -0.999944996144389,
    "x": 0.0,
    "y": 0.0,
    "z": -0.010488311865967603
  },
  "score": 0.8930596709251404,
  "timestamp": 315968385022970000,
  "track_label_uuid": null,
  "tracked": true,
  "width": 1.816713809967041
},
```

2. Data Association:

在所有 `lidar_timestamps` 反覆以下操作:

- (1) 先檢查一下資料(label, map 等等), 如遇到特定條件直接 continue 跳過, 進入下一個 `lidar_timestamps`
- (2) `ab3dmot.update` (在 city frame 座標系下更新觀測結果)
- (3) 整理結果(座標轉換等)並輸出 .json

其中, `ab3dmot.update` 包含以下操作:

- (1) 預測既有 `trackers` 的下一個 pose 和其他必要資訊
- (2) 將 `detections` 配對給 `trackers`。這是一個線性規劃問題, 使用 hungarian algorithm 求解。
- (3) 更新 `trackers` 的配對結果。
- (4) 幫沒有配對到的 `detections` 建立新的 `trackers`。
- (5) `Gaussian mixtures`

其中，trackers 使用 FilterPy 的 KalmanFilter 類別來實做。其系統建模類似於 constant velocity model。

- 假設 $t+1$ 時刻的真實狀態是從 t 時刻的狀態演化而來，符合下式：

$$\mathbf{x}_{t+1} = \begin{bmatrix} x + v_x \\ y + v_y \\ z + v_z \\ \theta \\ l \\ w \\ h \\ v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \theta \\ l \\ w \\ h \\ v_x \\ v_y \\ v_z \end{bmatrix} = F \mathbf{x}_t$$

- 時刻 $t+1$ ，對真實狀態 $\mathbf{x}(t)$ 的觀測滿足下式：

$$\mathbf{x}_{t+1} = \begin{bmatrix} x \\ y \\ z \\ \theta \\ l \\ w \\ h \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \theta \\ l \\ w \\ h \\ v_x \\ v_y \\ v_z \end{bmatrix} = H \mathbf{x}_t$$

另外，有別於 baseline code 的作法，GMF 幫 tracker 多定義了 `self.w`

- Each Gaussian component is weighted (`w_i`, [0,1]), weight is directly proportional to classification score and evolves according to it.

並使用以下參數來達成 **Gaussian mixtures**

- Probability of survival (`p_s` [0,1]) - probability of target survival between a consecutive time steps.
- Estimate threshold (`t_e` [0,1]) - threshold above which to report tracked targets.
- Prune threshold (`t_p` [0,1]) - threshold below which to remove Gaussian components from the tracker.

```

487 # KalmanBoxTracker init
488 # Test new
489 self.w = conf
490 self.ps = ps

496 #KalmanBoxTracker update
497 self.w += conf
498 if self.w > 1:
499     self.w = 1

550 #KalmanBoxTracker predict
551 # Multiply by p_s
552 self.w *= self.ps
553

747 #AB3DMOT update
748 for trk in reversed(self.trackers):
749     d = trk.get_state() # bbox location
750
751     if (trk.w >= self.thr_estimate):
752         ret.append(np.concatenate((d, [trk.id+1, trk.info])).reshape(1,-1))
753
754     i -= 1
755
756     if trk.w <= self.thr_prune:
757         self.trackers.pop(i)
758 
```

B. Contribution

1. 嘗試幫 KalmanFilter 指定不同的 Instance Variables

```
47 class KalmanBoxTracker(object):
48     """
49     This class represents the internal state of individual tracked objects observed as bbox.
50     """
51     count = 0
52     def __init__(self, bbox3D, info, p, conf, ps=1):
53         """
54         Initialises a tracker using initial bounding box.
55         """
56         self.kf = KalmanFilter(dim_x=10, dim_z=7)
57
58         # covariance matrix
59         self.kf.P = np.array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
60                                [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
61                                [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
62                                [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
63                                [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
64                                [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
65                                [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
66                                [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
67                                [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
68                                [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
69
70         # Process uncertainty/noise
71         self.kf.Q = np.array([[0.01, 0., 0., 0., 0., 0., 0., 0., 0., 0. ],
72                                [0., 0.01, 0., 0., 0., 0., 0., 0., 0., 0. ],
73                                [0., 0., 0.01, 0., 0., 0., 0., 0., 0., 0. ],
74                                [0., 0., 0., 0.01, 0., 0., 0., 0., 0., 0. ],
75                                [0., 0., 0., 0., 0.01, 0., 0., 0., 0., 0. ],
76                                [0., 0., 0., 0., 0., 0.01, 0., 0., 0., 0. ],
77                                [0., 0., 0., 0., 0., 0., 0.01, 0., 0., 0. ],
78                                [0., 0., 0., 0., 0., 0., 0., 0.01, 0., 0. ],
79                                [0., 0., 0., 0., 0., 0., 0., 0., 0.01, 0. ],
80                                [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.01]])
81
82         # measurement uncertainty/noise
83         self.kf.R = np.array([[0.1, 0., 0., 0., 0., 0., 0., 0. ],
84                                [0., 0.1, 0., 0., 0., 0., 0., 0. ],
85                                [0., 0., 0.1, 0., 0., 0., 0., 0. ],
86                                [0., 0., 0., 0.1, 0., 0., 0., 0. ],
87                                [0., 0., 0., 0., 0.1, 0., 0., 0. ],
88                                [0., 0., 0., 0., 0., 0.1, 0., 0. ],
89                                [0., 0., 0., 0., 0., 0., 0.1, 0. ]])
90
91         # measurement function
92         self.kf.H = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
93                                [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
94                                [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
95                                [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
96                                [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
97                                [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
98                                [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])
99
100         # state transition matrix
101         self.kf.F = np.array([[1., 0., 0., 0., 0., 0., 0., 0., 0.1, 0., 0. ],
102                                [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.1, 0. ],
103                                [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.1],
104                                [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0. ],
105                                [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0. ],
106                                [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0. ],
107                                [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0. ],
108                                [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0. ],
109                                [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0. ],
110                                [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.]])
111
```

2. 嘗試指定 > 1 的 α ，讓原本的 KalmanFilter 拓展成為 fading memory filter

```
131 # Assign a value > 1.0 to turn this into a fading memory filter.
132 self.kf.alpha = 1.05
```

3. Take measurements and predictions at steady state .但是似乎為錯誤用法，本系統非 LTI(線性非時變)系統。

```
173 # self.kf.update(bbox3D)
174 """
175 Add a new measurement (z) to the Kalman filter without recomputing the Kalman gain K, the state covariance P, or the system uncertainty S.
176
177 You can use this for LTI systems since the Kalman gain and covariance converge to a fixed value. Precompute these and assign them explicitly,
178 or run the Kalman filter using the normal predict()/update() cycle until they converge.
179
180 The main advantage of this call is speed. We do significantly less computation, notably avoiding a costly matrix inversion.
181
182 Use in conjunction with predict_steadystate(), otherwise P will grow without bound.
183 """
184 self.kf.update_steadystate(bbox3D)
```

```

198     # self.kf.predict()
199     """
200     Predict state (prior) using the Kalman filter state propagation equations. Only x is updated, P is left unchanged.
201     See update_steadystate() for a longer explanation of when to use this method.
202     """
203     self.kf.predict_steadystate()

```

4. 更換使用 Hungarian algorithm 呼叫的函式，需要再對 return 項調整一下形狀。

```

267     # matched_indices = linear_assignment(distance_matrix) # hungarian algorithm
268     matched_indices = linear_sum_assignment(distance_matrix)
269     matched_indices = np.asarray(matched_indices)
270     matched_indices = np.transpose(matched_indices)

```

5. 呼叫 argoverse-api 裡的函式，對 map 多檢查兩個條件（這原本就有，我只是把註解打開）

```

179     # Check the driveable/roi area
180     da = am.remove_non_driveable_area_points(np.array([ego_xyz]), city_name=city_name)
181     if len(da) == 0 and l['label_class'] == 'VEHICLE':
182         continue
183
184     roi = am.remove_non_roi_points(np.array([ego_xyz]), city_name=city_name)
185     if len(roi) == 0:
186         continue

```

6. 做關聯性表現衡量時，計算 match-distance 的方式由 IOU 改成 Mahalanobis distance（這也是本來就寫在裡面的選項）

7. GMF 的作者在它的 README 裡面寫著，可以根據不同性能要求調整這兩個參數

```

280     thr_estimate = 0.55
281     ps = 0.875

```

C. Problem and Solution

1. Kalman Filter 的預設值不一定會 work

- 狀態轉移矩陣(F 矩陣)右上角需根據資料的時間變化量(SEGMENT DURATION)做調整。
- uncertainty(Q 矩陣,R 矩陣)需要靠經驗來嘗試調整。

2. 加了 alpha 之後性能變差

覺得只是值給的不好。原 baseline code 的作者在他們的 paper 中提到，類似於 constant velocity model 的 KalmanFilter，在轉彎上特別不行。fading memory filter 用遞迴的方式處理更多歷史資料，應該能協助我們克服此系統建模上的瑕疵。繼續嘗試給它不同的 scale (alpha)或許能得到更好的結果。

3. 不同的線性規劃函式差異

sklearn.utils.linear_assignment_ 和 scipy.optimize.linear_sum_assignment 實做方式不同。
參考資料：<https://github.com/scikit-learn/scikit-learn/issues/13464>



yiaikwy commented on Mar 10, 2020 · edited

@jnhansen @praths007 I am concerned that `linear_sum_assignment` is not equivalent to `linear_assignment` which later implements "maximum values" matching strategy not "complete matching" strategy, i.e. in tracking problem maybe an old landmark lost and a new detection coming in. We don't have to make a complete assignment, just match as more as possible.

I also try to tell others that it is dangerous to use `linear_sum_assignment` because it implements complete matching, where in the most cases we don't need.

Instead of using complete matching by KM algorithms, mini cut - max flow might be a good choice.

D. Results

在固定 $\text{thr_estimate} = 0.5$, $\text{ps} = 0.875$, 使用原本未經修改的 GMF 得到最好的結果。

C:MOTA	P:MOTA	C:MOTPD	P:MOTPD	C:MOTPO	P:MOTPO	C:MOTPI	P:MOTPI	C:IDF1	P:IDF1	C:MT	P:MT	C:ML	P:ML	C:FP	P:FP	C:FN	P:FN	C:SW	P:SW	C:FRG	P:FRG
68.24	44.16	0.38	0.25	52.66	83.16	0.21	0.29	0.82	0.59	0.56	0.29	0.18	0.3	13244	4160	18790	13193	136	184	733	370

E. Run the code

需修改 `run_ab3dmot_GMF.py` 以下三行路徑

```
261 path_detections = f"/home/yellow/self-driving-cars-course/final/argoverse_detections_2020/validation"
262 path_data = f"/home/yellow/self-driving-cars-course/final/argoverse_tracking/val"
263 path_results = f"/home/yellow/self-driving-cars-course/final/argoverse_cbgs_kf_tracker/results/results_tracking_val_cbgs"
```

終端機執行 `python3 run_ab3dmot_GMF.py`