

✓ 卡尔曼滤波

✎ 团战：A说张良没大；B说谁去骗张良大；我该怎么选呢？

✎ 那就看我能更信这两人当中哪一个了（谁的权重大）

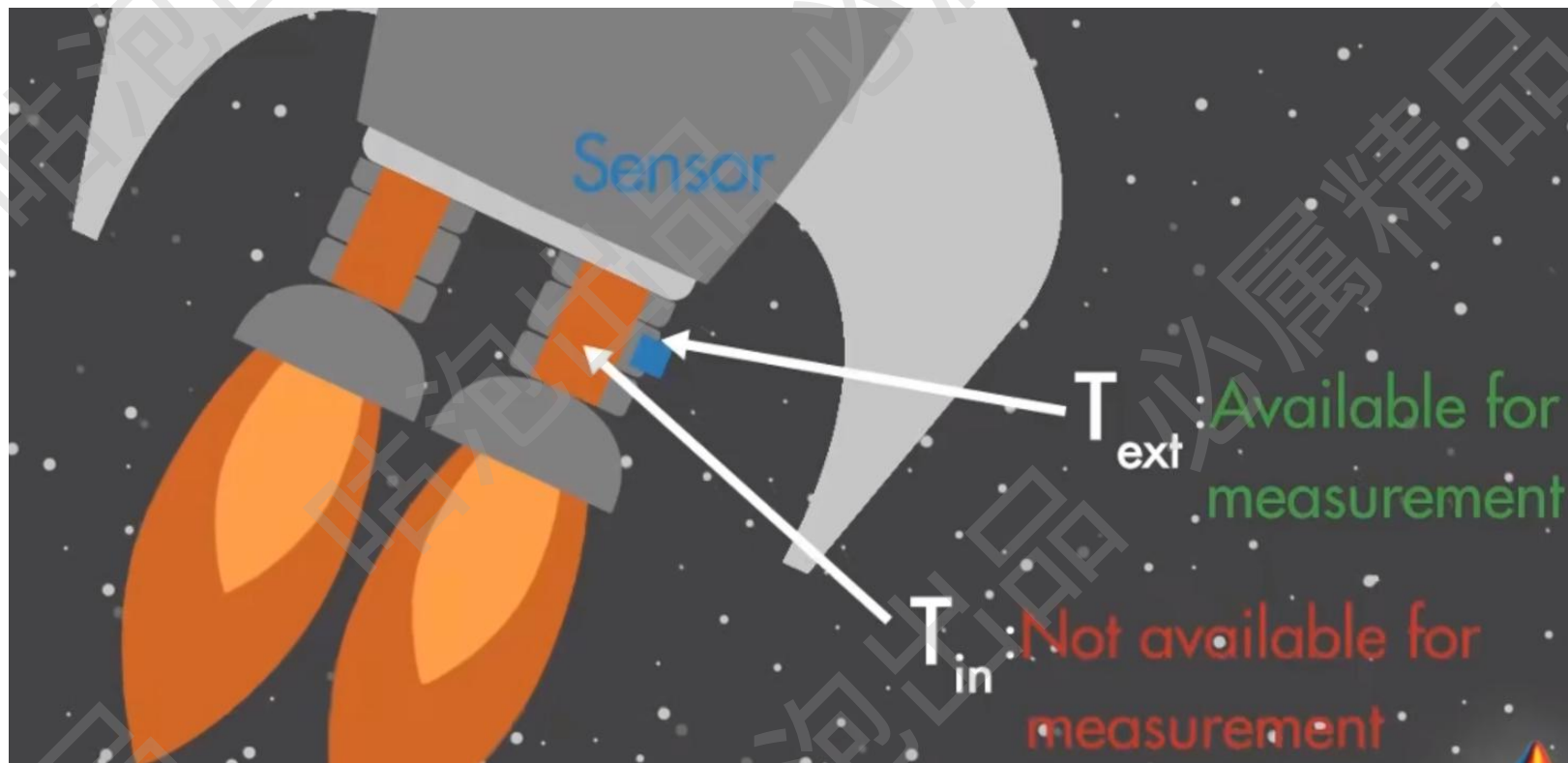
✎ 但是现在只有观测值（A说张良没大）和一个估计值（数学模型预测张良CD）

✎ 这个时候我该怎么选？如何选才能得到更优的结果呢？

deepsort

✓ 卡尔曼滤波

✎ 要测量外部温度，但是传感器只能放在内部，基于内部结果进行外部估计



✓ 卡尔曼滤波

✎ 开车时，如何准备定位自己的位置呢？

(1.提供加速度信息； 2.里程表等信息； 3.GPS信息；但他们都有误差)



Inertial measurement unit (IMU)
measures acceleration and
angular velocity of the car.



Odometer provides the
relative position of the car.



GPS receiver provides the
absolute position of the car.

deepsort

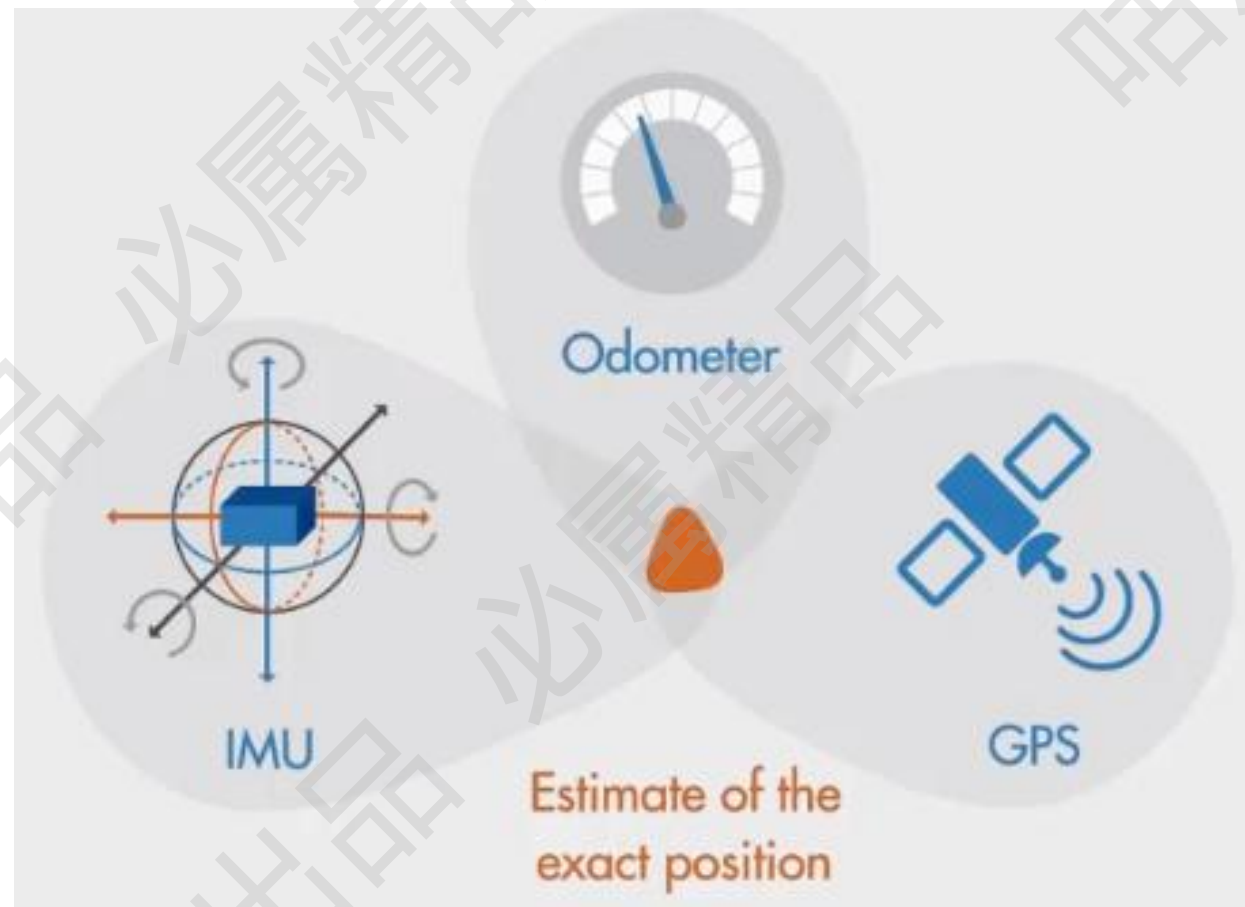
✓ 卡尔曼滤波

✎ 结合已知信息估计最优位置

✎ 本质是优化估计算法

✎ 例如估计人在下一帧的位置

✎ 阿波罗登月这哥们也用上了



✓ 卡尔曼滤波

✎ 一个小车的例子，位置的更新：

状态向量：(位置和速度) $\mathbf{x}_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix}$

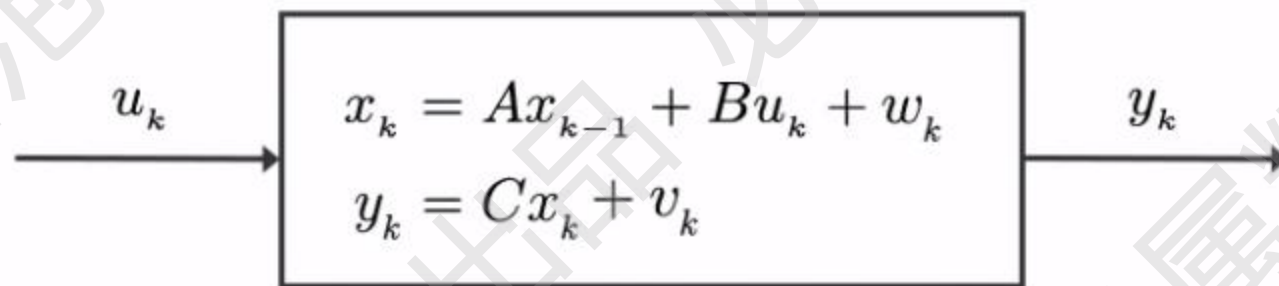
小车的加速度： $\mathbf{u}_t = \frac{f_t}{m}$

下一时刻位置：
$$\begin{cases} x_t = x_{t-1} + \dot{x}_{t-1} \Delta t + \frac{1}{2} \frac{f_t}{m} \Delta t^2 \\ \dot{x}_t = \dot{x}_{t-1} + \frac{f_t}{m} \Delta t \end{cases}$$

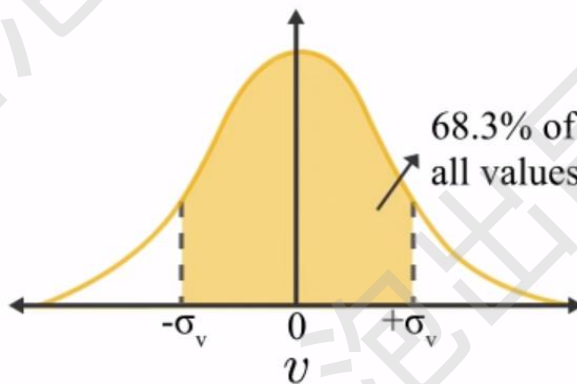
矩阵表达式：
$$\begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ \dot{x}_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} \frac{f_t}{m}$$
 其中： $\mathbf{F}_t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$, $\mathbf{B}_t = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}$

✓ 卡尔曼滤波

✎ 任何状态都会受外部环境的影响（例如车压了块石头），通常呈正态分布

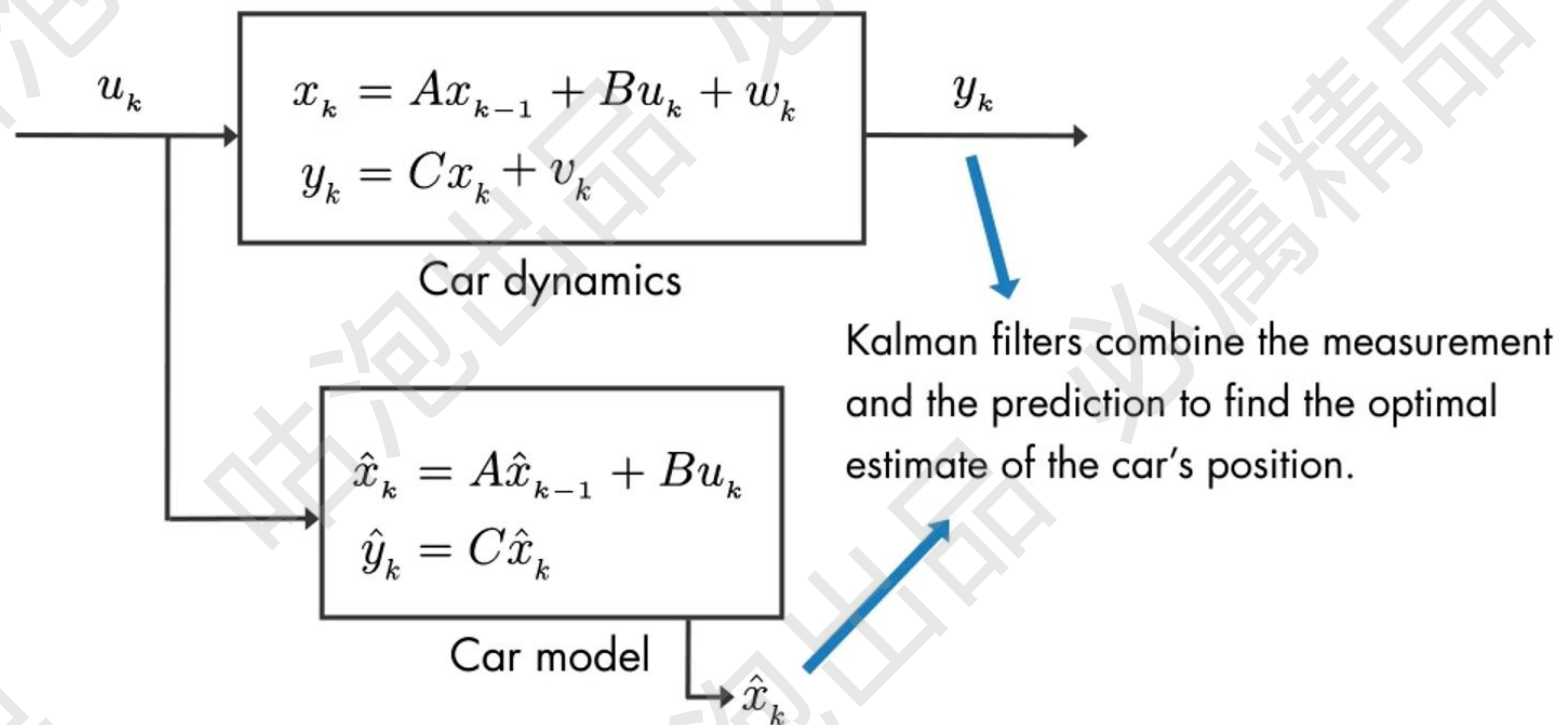


Car dynamics



✓ 卡尔曼滤波

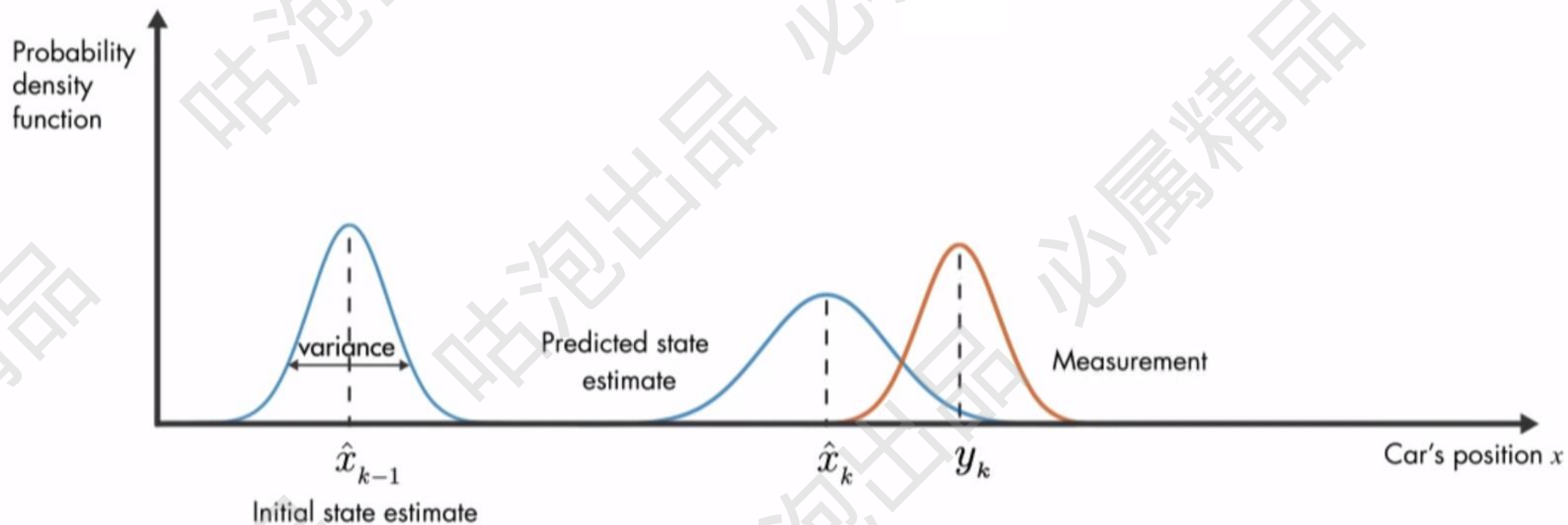
📌 本质上就是基于估计值和观测值进行综合（如下一帧预测值和下一帧检测值）



deepsort

✓ 卡尔曼滤波

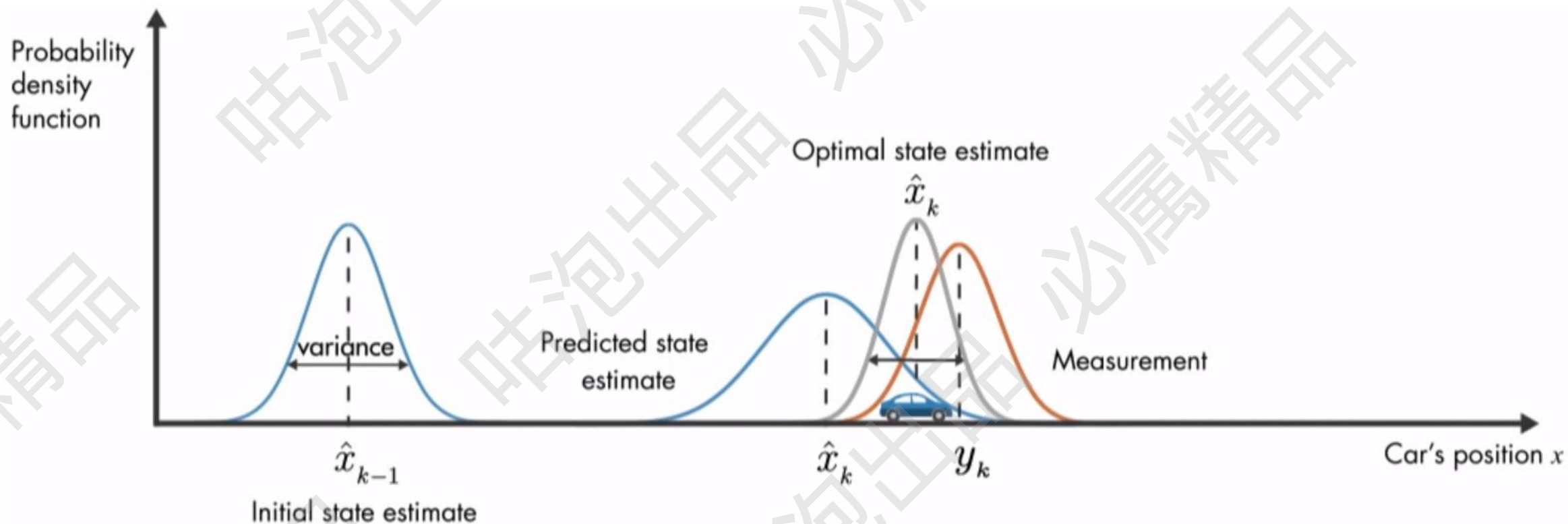
📎 下一时刻，最准确的位置在哪呢？(\hat{x} 是预测值， y 是观测值)



deepsort

✓ 卡尔曼滤波

✎ 其实就是两个分布的乘积，得到更准确的估计（像是取交集）



deepsort

✓ 卡尔曼滤波

✎ \hat{x}^- : 先验估计

✎ \hat{x} : 后验估计 (用到观测值进行修正)

$$\hat{x}_k = \underbrace{A\hat{x}_{k-1} + Bu_k}_{\hat{x}_k^-} + K_k(y_k - \underbrace{C(A\hat{x}_{k-1} + Bu_k)}_{\hat{x}_k^-})$$

A Posteriori Estimate

$$\hat{x}_k = \underbrace{\hat{x}_k^-}_{\text{Predict}} + \underbrace{K_k(y_k - C\hat{x}_k^-)}_{\text{Update}}$$

deepsort

✓ 卡尔曼滤波

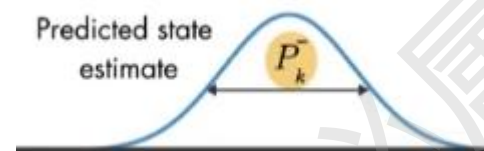
📌 两大核心模块：

Prediction
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
$P_k^- = AP_{k-1}A^T + Q$

Update
$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R}$
$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$
$P_k = (I - K_k C)P_k^-$

📌 预测阶段：预测状态估计值及其协方差

📌 在单状态中，协方差矩阵就是其方差：



📌 它就是预测状态中的不确定性的度量（噪声导致）

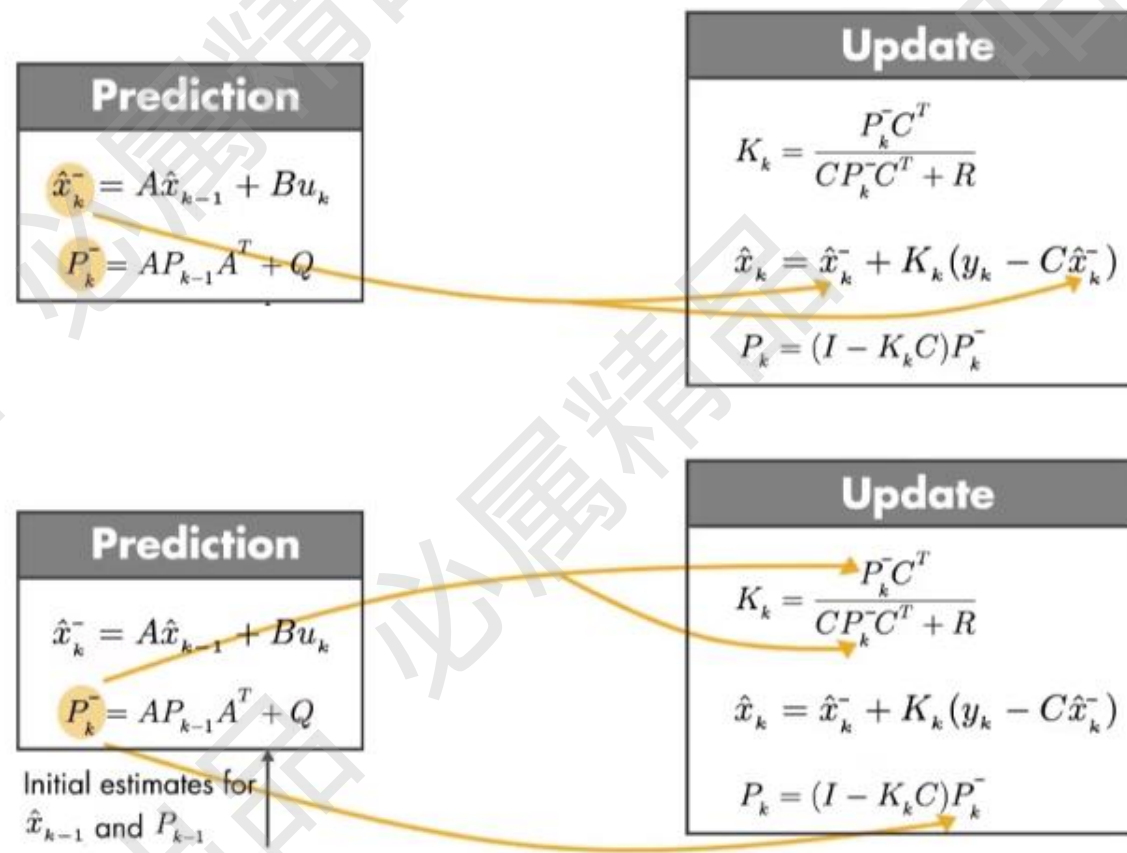
✓ 卡尔曼滤波

✎ 第二步要基于预测值更新参数

✎ 例如追踪每一帧的状态肯定要变的

✎ 预测完需要根据观测值来修正

✎ 修正后的状态值去估计下一帧



deepsort

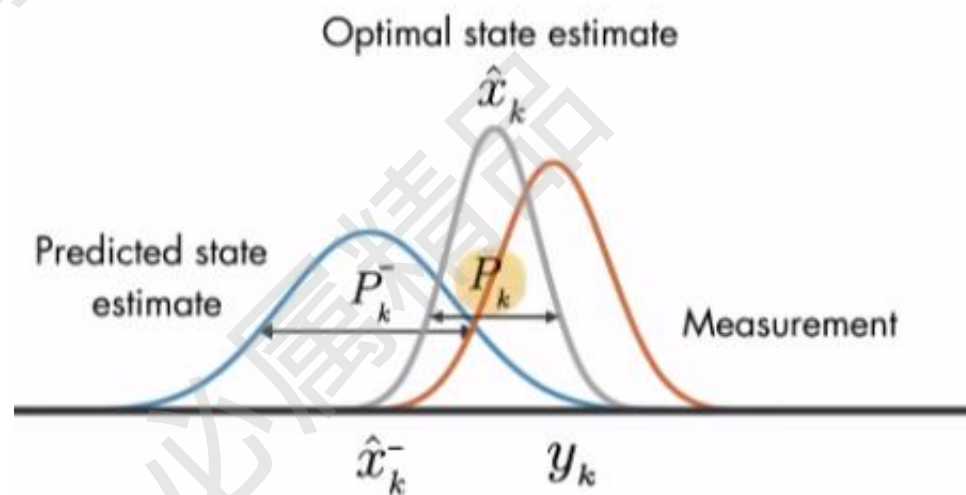
✓ 卡尔曼滤波

✎ 卡尔曼增益K: $K_k = \frac{P_k^- C^T}{C P_k^- C^T + R}$

✎ 它的目的就是让最优估计值的方差更小

✎ 相当于一个权重项，该怎么利用估计与观测

✎ 它就决定了卡尔曼滤波的核心作用



✓ 卡尔曼滤波

✎ 当观测噪音没有时

✎ 增益K化简后结果:

✎ 相当于最优估计等于观测

✎ 其中C表示转换矩阵
(单状态, 匀速)

Calculation of \hat{x}_k

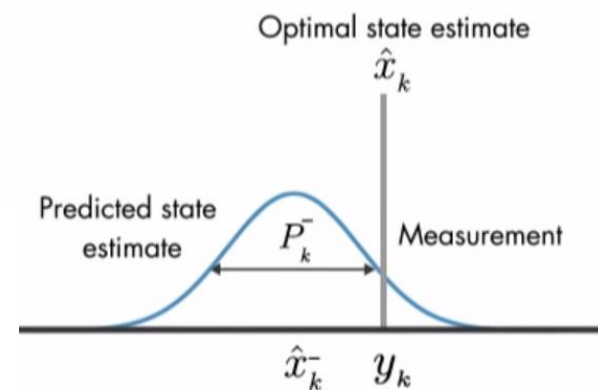


$$\lim_{R \rightarrow 0} K_k = \lim_{R \rightarrow 0} \frac{P_k^- C^T}{C P_k^- C^T + R} = \lim_{R \rightarrow 0} \frac{P_k^- C^T}{C P_k^- C^T + 0} = C^{-1} \quad C^{-1} = 1$$

$$\hat{x}_k = \hat{x}_k^- + K_k (y_k - C \hat{x}_k^-) = \hat{x}_k^- + C^{-1} (y_k - C \hat{x}_k^-)$$

$$= \cancel{\hat{x}_k^-} + C^{-1} y_k + \cancel{C^{-1} C \hat{x}_k^-}$$

$$\hat{x}_k = y_k$$



✓ 卡尔曼滤波

✎ 当状态估计没有噪音时

✎ 增益K化简后结果:

✎ 相当于最优估计等于预测值

✎ 其中C表示转换矩阵
(单状态, 匀速)

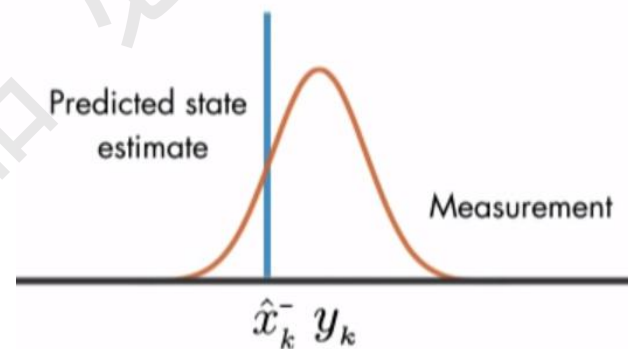
Calculation of \hat{x}_k



$$\lim_{P_k^- \rightarrow 0} K_k = \lim_{P_k^- \rightarrow 0} \frac{P_k^- C^T}{C P_k^- C^T + R} = \lim_{P_k^- \rightarrow 0} \frac{0}{0 + R} = 0$$

$$\hat{x}_k = \hat{x}_k^- + K_k (y_k - C \hat{x}_k^-) = \hat{x}_k^- + 0(y_k - C \hat{x}_k^-)$$

$$\hat{x}_k = \hat{x}_k^-$$



deepsort

✓ 卡尔曼滤波

✎ 追踪问题需要考虑的状态:

✎ 均值(Mean): 8维向量表示为 $x = [cx, cy, r, h, vx, vy, vr, vh]$

✎ 中心坐标 (cx, cy) , 宽高比 r , 高 h , 以及各自的速度变化值组成

✎ 协方差矩阵: 表示目标位置信息的不确定性, 由 8×8 的矩阵表示

deepsort

✓ 卡尔曼滤波

✎ 追踪过程也要分为两个阶段：

✎ 每一个track都要预测下一时刻的状态，并基于检测到的结果来修正
(匀速，线性，咱们追踪通常都是一帧一帧处理的)

$$\underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}}_{x'}_{t+1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_F \cdot \underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}}_x_t$$

deepsort

✓ 匈牙利算法

- ✎ 完成匹配的同时最小化代价矩阵（田忌赛马？）
- ✎ 当前帧检测到的目标该匹配到前面哪一个track呢？
- ✎ 感觉有时候并不是最优分配，而是尽可能多的分配
- ✎ 目标追踪任务，detr目标检测任务中都用到了该方法

deepsort

✓ 匈牙利算法小例子

✎ 给3个人分配3个任务，代价矩阵如下（就当花的时间吧）

✎ 哪个人做哪个任务能使得代价矩阵最小呢？

	Task1	Task2	Task3
Person1	15	40	45
Person2	20	60	35
Person3	20	40	24

✓ 匈牙利算法小李子

- ✎ 如果代价矩阵的某一行或某一列同时加上或减去某个数，则这个新的代价矩阵的最优分配仍然是原代价矩阵的最优分配。
- ✎
 1. 对于矩阵的每一行，减去其中最小的元素
 2. 对于矩阵的每一列，减去其中最小的元素
 3. 用最少的水平线或垂直线覆盖矩阵中所有的0
 4. 如果线的数量 = N ，则找到了最优分配，算法结束，否则进入5
 5. 找到没有被任何线覆盖的最小元素，每个没被线覆盖的行减去这个元素，每个被线覆盖的列加上这个元素，返回步骤3

deepsort

✓ 匈牙利算法小例子

✎ 每一行最小的元素分别为15、20、20，减去得到：

	Task1	Task2	Task3
Person1	0	25	30
Person2	0	40	15
Person3	0	20	5

✎ 每一列最小的元素分别为0、20、5，减去得到：

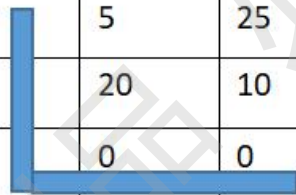
	Task1	Task2	Task3
Person1	0	5	25
Person2	0	20	10
Person3	0	0	0

deepsort

✓ 匈牙利算法小例子

✎ 用最少的水平线或垂直线覆盖所有的0，得到：

	Task1	Task2	Task3
Person1	0	5	25
Person2	0	20	10
Person3	0	0	0

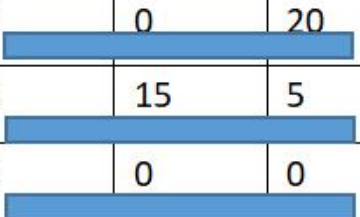


✎ 线的数量为2，小于3，进入下一步，没被覆盖的最小元素是5，没被覆盖的行（第一和第二行）减去5；被覆盖的列（第一列）加上5，结果如下：

	Task1	Task2	Task3
Person1	-5	0	20
Person2	-5	15	5
Person3	0	0	0

	Task1	Task2	Task3
Person1	0	0	20
Person2	0	15	5
Person3	5	0	0

	Task1	Task2	Task3
Person1	0	0	20
Person2	0	15	5
Person3	5	0	0



deepsort

✓ 匈牙利算法

✎ (T2, P1) (T1, P2) (T3, P3) 分配时代价最小

	Task1	Task2	Task3
Person1	0	0	20
Person2	0	15	5
Person3	5	0	0

	Task1	Task2	Task3
Person1	15	40	45
Person2	20	60	35
Person3	20	40	25

✎ sklearn: linear_assignment(); scipy: linear_sum_assignment()

✎ 运动信息匹配 (卡尔曼估计) ; 外观匹配 (ReID) ; IOU匹配 (BBOX)

✓ ReID特征

✎ 追踪人所以用到了ReID，如果追踪其他目标需要自己训练

✎ 很简单的一个网络结构，对输入的bbox进行特征提取，返回128维特征

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

deepsort

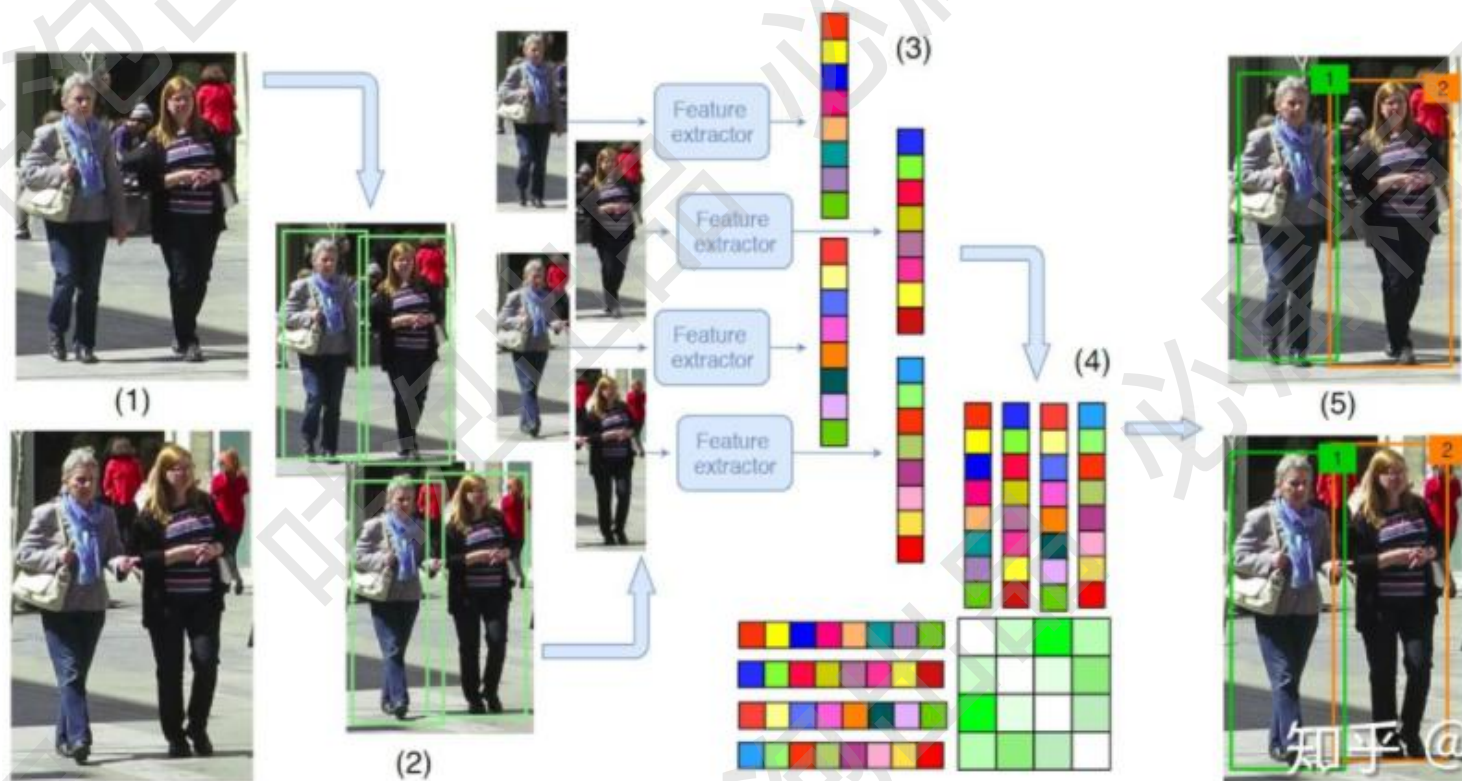
✓ ReID特征

- ✎ 根据当前检测的所有bbox与当前所有track，先得到其所有ReID特征
- ✎ 当前每一个track均存了一个特征序列（就是每一次匹配都会保留一份特征）
- ✎ 例如一个track有5份128维向量，选其与每个bbox余弦距离最小的作为输入
- ✎ track保存的特征数量是有上限的，默认参数是100个

deepsort

✓ 追踪任务基本流程

📎 目标检测+追踪 (对检测的bbox提取各项特征后进行匹配)



deepsort

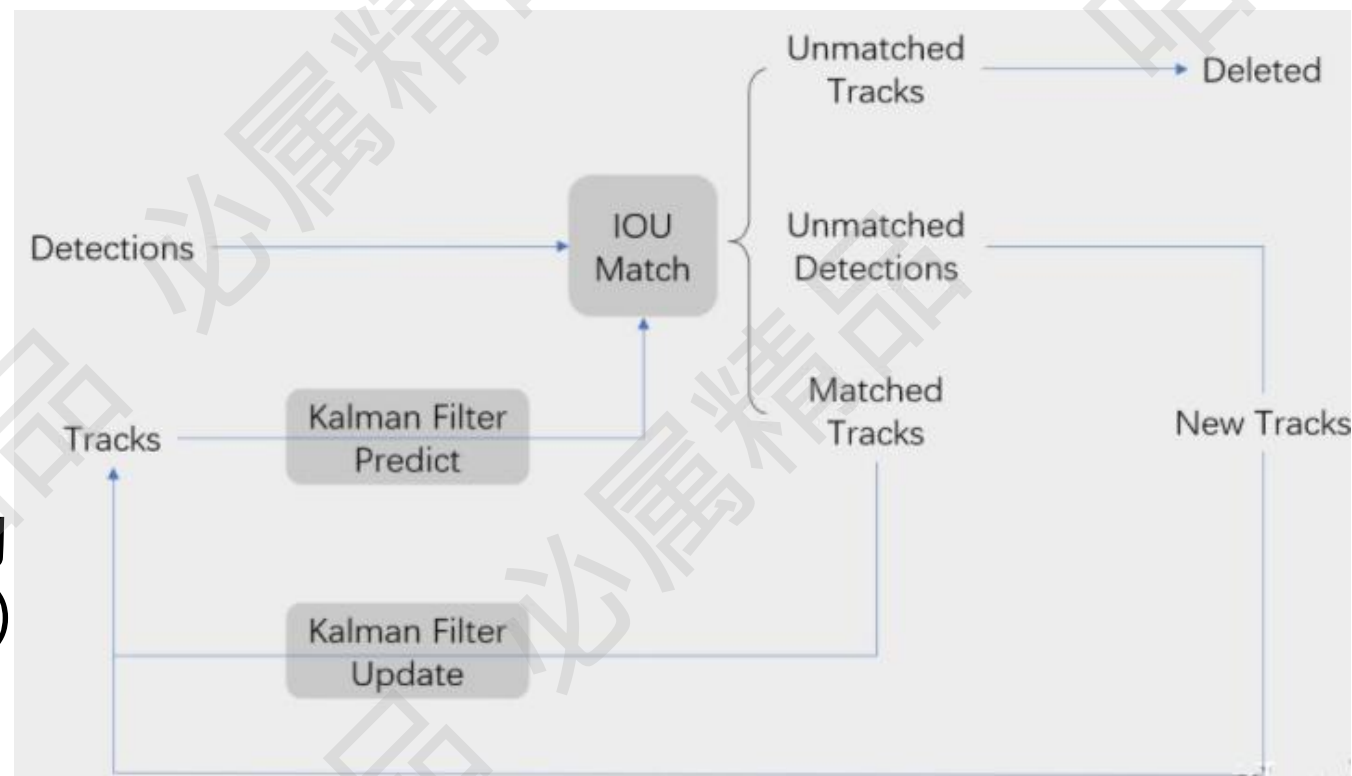
✓ sort算法

✎ 1.卡尔曼预测与更新

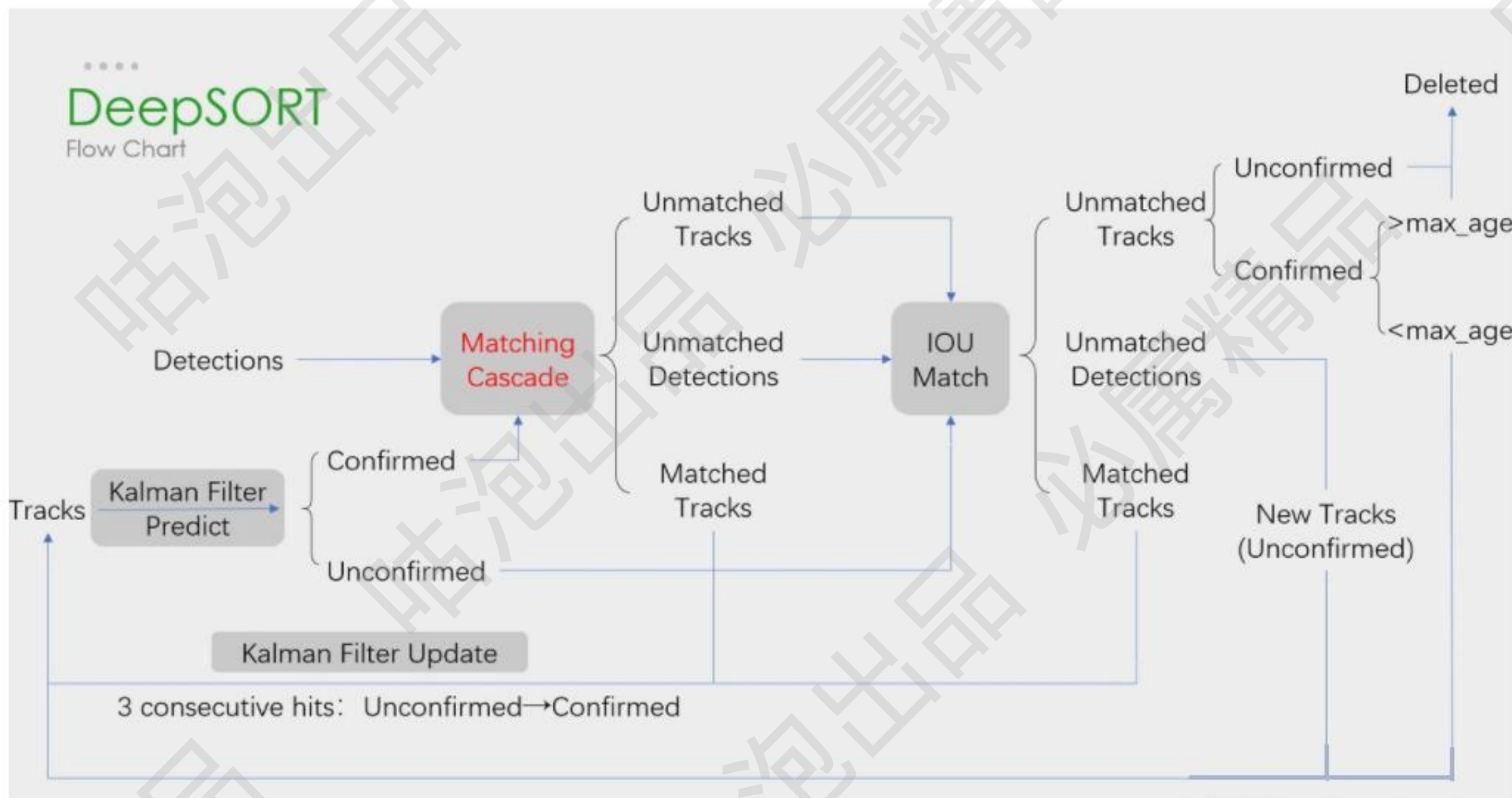
✎ 2.匈牙利匹配返回结果

✎ 将预测后的tracks和当前帧中的detections进行匹配 (IOU匹配)

✎ 木有REID等深度学习特征



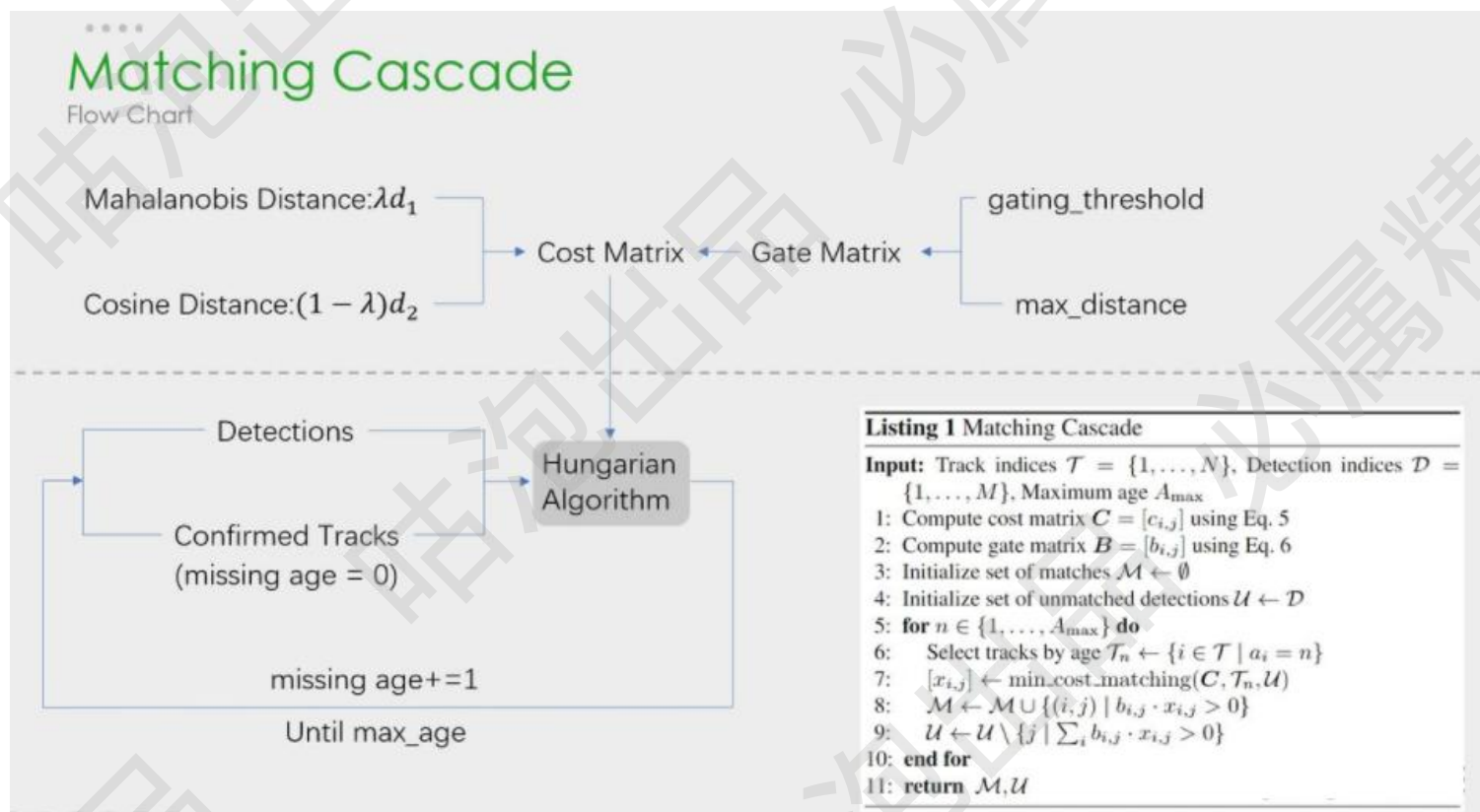
✓ deepsort算法



deepsort

✓ deepsort算法

📎 级联匹配：代码函数由运动特征(卡尔曼预测)与ReID特征计算的距离组成



deepsort

✓ deepsort算法

✎ 级联匹配是一个循环，大概是这个意思

✎ 追踪过程中肯定有些track会丢失目标，比如missing age=20，丢了二十帧

✎ 匹配过程中会先从missing age=0开始，也就是没丢过的先匹配，然后按顺序

✎ 丢失最多的最后匹配，参数中默认上限是70

deepsort

✓ deepsort算法

✎ 在选择距离权重时，也可以将 $\lambda = 0$ ，因为卡尔曼预测基于匀速模型



✎ 如果非匀速模型则不适用；阈值就相当于一个门，由于匹配会尽可能多

✎ 有时候差异太大的时候，那就别匹配了，会设置成一个比较大的数

deepsort

✓ 追踪流程拆解

✎ 第一帧：检测得到10个bbox，此时木有track

✎ （对每一帧的检测结果，都会经过NMS和置信度阈值来筛选）

✎ 不会执行任何deepsort追踪操作（还木有track）

✎ 对当前10个检测结果分别初始化track

deepsort

✓ 追踪流程拆解

- ✎ 第二帧：检测得到11个bbox，如何跟上一帧初始化的track匹配呢？
- ✎ 进行IOU匹配（级联匹配需要对confirmed track，现在还木有）
- ✎ 此时匹配到10个track，注意匹配到的别忘了更新其卡尔曼参数
- ✎ 还有一个bbox没有匹配上此时为其新建一个track

deepsort

✓ 追踪流程拆解

- ✎ 第三帧：检测得到12个bbox，当前还没有confirm的track（命中3次）
- ✎ 前面创建了11个track要继续与12个bbox进行IOU匹配
- ✎ 其中一个track没有匹配上（此时其是非confirm的，将其删除）
- ✎ 有10个track已经命中3次了，将其状态更改为confirm

deepsort

✓ 追踪流程拆解

✎ 第四帧：检测得到14个bbox，不仅要做法IOU还要做法级联匹配

✎ 对前面10个confirm的track进行级联匹配（优先）然后再IOU

✎ 只有confirm的track才会可视化在输出结果中

✎ 注意每次匹配到的track一定要更新其卡夫曼参数

✓ 追踪流程拆解总结

- ✎ 1.检测得到当前帧的bbox（其实追踪好坏主要取决于检测结果。。。）
- ✎ 2.track分为：confirmed和unconfirmed它俩待遇不同
- ✎ 3.对于confirmed要先进行级联匹配，它们优先级高，连续70帧没匹配上会删除
- ✎ 代价矩阵包括Reid特征构建的余弦距离与运动信息构建的马氏距离

✓ 追踪流程拆解总结

- ✎ 4.对于级联匹配玩剩下的和当前是unconfirmed与剩下的bbox进行IOU匹配
- ✎ 5.经过级联与IOU匹配后就得到了所有匹配结果，别忘了更新track参数
- ✎ 对于匹配成功的track，连续命中3次以上才能转换成confirmed
- ✎ 对于没有匹配成功的track，如果它是unconfirmed则删除