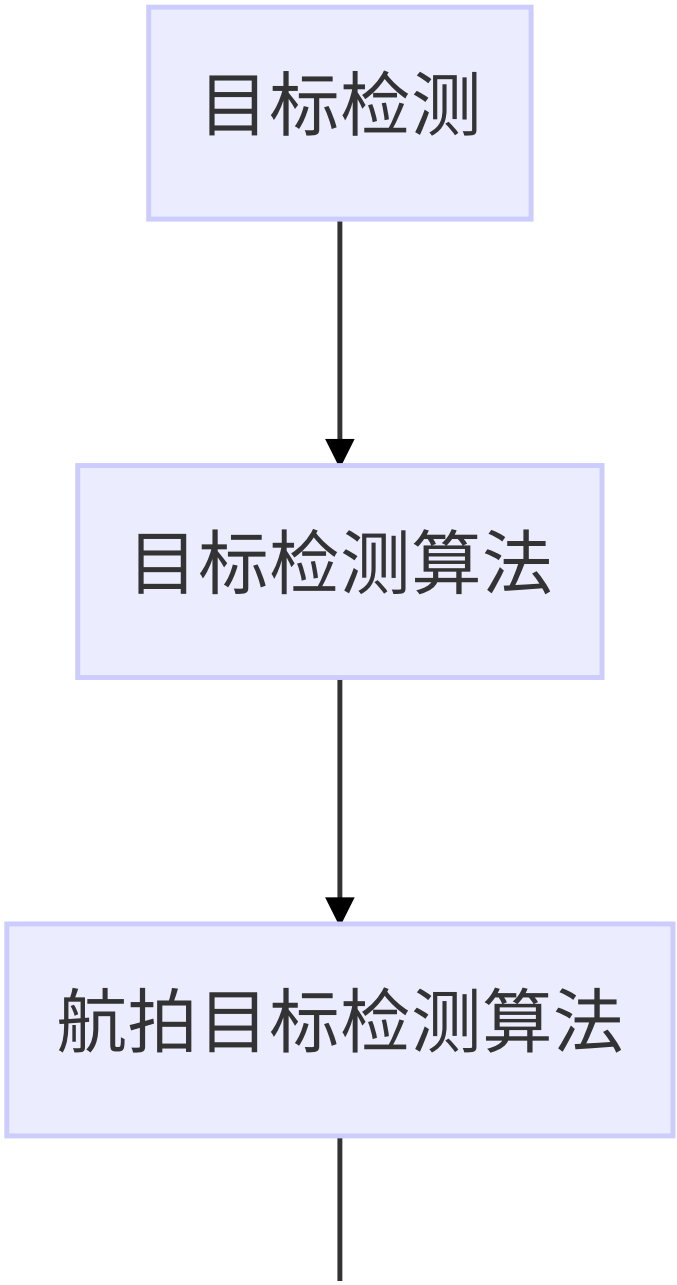
航拍追车算法调研

1 目的

设计一种追车算法，适合 Falcon 跟车。

2 概述

调研流程如下



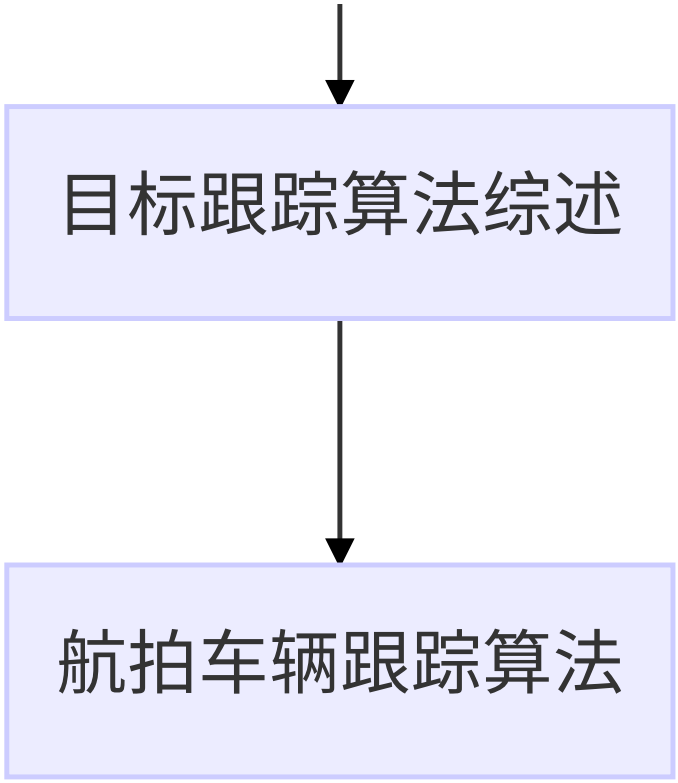


图1 航拍追车算法调研流程图

3 目标检测

考虑项目以深度学习的目标检测算法为基础，故按照有无 Regional Proposal主要分两大类：

- Regional Proposal：代表 Faster R-CNN。**优点是检测精度高，缺点是检测速度慢**；
- 回归：代表 YOLO，**优点是检测速度快，但检测精度低**。

4 目标检测算法

4.1 Region Proposal方法

根据[1]，原理是利用图像颜色，纹理，边缘等信息预先找到图像中目标可能出现的位置；选取较少候选窗口，该窗口可多种规模，能达到滑动窗口一样的召回率，降低计算复杂度；实施目标检测。

代表算法：

1. R-CNN： 利用 Selective Search算法，在一张图像中提取 2000哥候选区域，每个候选区域缩放为227*227。然后卷积提取特征，将特征输入SVM分类。数据集采用 PASCAL VOC2007，多标签图像分类任务[3]的mean average precision 值（模型好坏评估的指标[2]) 可知，模型准确率从34.3%提升到 66%[4]。由于使用大量卷积操作，检测耗时严重；
2. SPP-NET：针对 CNN问题，SPP-NET先对图像提一次卷积特征，然后将区域推荐框在原图的位置映射到卷积层特征图上，接着对每个特征图的区域特征采用变化的下采样处理方式。简言之，将任

意尺度特特征图映射到同样唯独，使不同尺寸的区域候选经过映射后，变成统一尺寸，即可作为同一全连接层的输入；
3. Fast R-CNN：融合 R-CNN和SPP-NET优点，使用一层 ROI 池化层下采样得到 7*7的特征图，并用Softmax代替 SVM分类，引入边框回归损失函数。PASCAL VOC2007的检测 mAP提升到了 66.9% [5]，每张图片检测时间降低3秒；
4. Faster R-CNN：采用 Region Proposal Network, RPN, 代替 Selective Search 来提取区域候选框， RPN 和 Fast R-CNN共享卷积网络，区域候选框由卷积神经网络直接产生，PASCAL VOC2007上mAP值达到 73.2%， 每张图片检测时间缩短到 200 ms.

4.2 回归方法

原理是给定输入图像，直接在图像上多个位置回归出该位置的目标边框和目标类别。

代表算法：

1. YOLO： 首先将图像划分为 7*7的网格，每个网格预测两个检测框，每个检测框包含各自位置信息、框中目标的置信度、以及检测区域在预测类别上的概率，然后利用阈值去掉置信度较低的目标检测框，根据极大值抑制法(NMS)去除冗余的检测框。同样数据集，mAP值63.4%，检测速度 45 FPS[7]；

- SSD：结合 YOLO回归思想和 Faster R-CNN的 anchor机制，使用全图的各个卷积层的多尺度区域特征进行回归，平衡精度和速度的效果。同样数据集，mAP值为 72.1%，速度 58 FPS[8]。

5 航拍目标检测算法

在国外，以上基于深度学习的通用目标检测算法都是针对 PASCAL VO2007, ImageNet等公开数据集，该数据集特点是目标物体在图像中所占面积较大，包含目标信息较多，经过卷积神经网络能够提取很多特征。如果是航拍图像，通用算法效果不好，无法实用。文献[9]后来采用若监督学习的车辆检测算法，利用图像及标注数据度检测网络的前几层卷积层进行训练，再利用少量的检测框级的标注数据对模型进行微调，从而得到高效的检测模型。文献[10]基于航拍图像快速统计车辆数方法。利用浅层的深度学习网络提取图像的特征，采用检测器对图像中车辆进行检索，再利用分类器对检测框进行分类，从而用回归器调整检测框的位置并对车辆的车头朝向进行预测。然而该方法假设飞行器的拍摄高度和视角都是固定的，算法没有鲁棒性。

在国内，大部分车辆检测针对场景是大尺寸车辆的图像，而非航拍图像，例如百度，商汤尅及，Face++， Link Face等。文献[11]基于公路视频实时车流检测，主要是背景自动更新算法。算法由背景帧差和自动更新背景两部分组成。该算法假设背景不变，一旦改变，现实场景检测效果不好。文献[1]针对 Munich 航拍数据集和自身课题数据集（高度90-150米，角度固定为30度，45度，60度，90度），提出改进的YOLO模型，mAP值高达 75.7%，不同图像尺寸下416、480、544、608时，算法检测耗时均在50ms以下，最快可有 1 ms；

6 目标跟踪算法综述

6.1 近两年概况

目标跟踪算法，2017年和2018年已公开的CVPR论文参考以下链接

- https://www.zhihu.com/question/26493945
- https://blog.csdn.net/weixin_40645129/article/details/81173088

为防止有误，建议参考以下官网

- http://cvpr2017.thecvf.com/
- http://cvpr2018.thecvf.com/

6.2 开源算法

首先，开源算法在此处定义为网上开源或从作者那里可得到的跟踪算法。

其次，考虑我司项目目标是单目标或多目标车辆跟踪，故忽略航拍车流跟踪算法类的调研。例如，文献[1]中作者采用匹配式追踪方法，将前后两帧检测结果作为模型的输入，根据检测框尺寸，图像颜色直方图，位置等信息进行两帧间检测狂的匹配，从而实现对车辆的追踪。

下面将专注于单/多目标追车算法调研。根据2013年的文献[12]

表1 - 在线跟踪算法综述

序号	算法名称	模型原理	检测模块	在线模型是否更新	编程语言	帧率FPS

序号	算法名称	模型原理	检测模块	在线模型是否更新	编程语言	帧率FPS
1	CPF	local, intensity histogram	particle filter	否	C	109
2	LOT	local, color	particle filter	是	Matlab	0.7
3	IVT	holistic, PCA, generative model	particle filter	是	Matlab+C/C++	33.4
4	ASLA	local, SPCA, generative model, discriminative model	particle filter	是	Matlab+C/C++	8.5
5	SCM	holistic, sparse representation, generative model, discriminative model	particle filter	是	Matlab+C/C++	0.51
6	L1APG	holistic, spare representation, generative model	particle filter	是	Matlab+C/C++	2
7	MTT	holistic, sparse representation, generative model	particle filter	是	Matlab	1
8	VTD	holistic, SPCA, generative model	Markov Chain Monte Carlo	是	可执行文件+Matlab+C/C++	5.7
9	VTS	local, SPCA, generative model	Markov Chain Monte Carlo	是	可执行文件+Matlab+C/C++	5.7
10	LSK	local, sparse representation, generative model	local optimum search	是	可执行文件+Matlab	5.5
11	ORIA	holistic, template, generative model	local optimum search	是	Matlab	9

序号	算法名称	模型原理	检测模块	在线模型是否更新	编程语言	帧率FPS
12	DFT	local, template	local optimum search	是	Matlab	13.2
13	KMS	holistic, intensity histogram	local optimum search	否	C	3.159
14	SMS	holistic, intensity histogram	local optimum search	否	C	19.2
15	VR-V	holistic, color	local optimum search	是	Matlab+C/C++	109
16	Frag	local, intensity histogram	dense sampling search	否	C	6.3
17	OAB	holistic, haar, discriminative model	dense sampling search	是	C	22.4
18	SemiT	holistic, haar, discriminative model	dense sampling search	是	C	11.2
19	BSBT	holistic, haar, discriminative model	dense sampling search	是	C	7
20	MIL	holistic, haar, discriminative model	dense sampling search	是	C	38.1
21	CT	holistic, haar, discriminative model	dense sampling search	是	Matlab+C/C++	64.4
22	TLD	local, binary pattern, discriminative model	dense sampling search	是	Matlab+C/C++	28.1

序号	算法名称	模型原理	检测模块	在线模型是否更新	编程语言	帧率FPS
23	Struck	holistic, template, discriminative model	dense sampling search	是	C	20.2
24	CSK	holistic, template, discriminative model	dense sampling search	是	Matlab	362
25	CXT	holistic, binary patter, discriminative model	dense sampling search	是	C	15.3

根据初始化准则可分为三种测试方法：

- One Pass Evaluation-**OPE**: evaluate one tracker on the entire sequence with initialization from the ground truth position in the first frame;
- temporal robustness evaluation-**TRE**: sampling initialization temporally , i.e. start at different frames;
- spatial robustness evaluation-**SRE**: sampling initialization spatially, i.e. start by different bounding boxes in initialized frame.

根据性能评估准则可分为两种标准：

- Precision Plots[13]: the center location error which is defined as the Euclidean distance between the center locations of the tracked target and the manually labeled ground truth. Then the average center location error over all the frames of one sequence is used to summarize the overall performance for that sequence. However, when the tracker loses the target, the output bounding box can be random so that sometimes the average error value may not measure the tracking performance correctly;
- Success Plots: calculate intersection of $T_1 \cap T_2$, then divided by their union $T_1 \cup T_2$. If this value is bigger than a given threshold, then tracker performance is ranked by their result.

最后整体评估

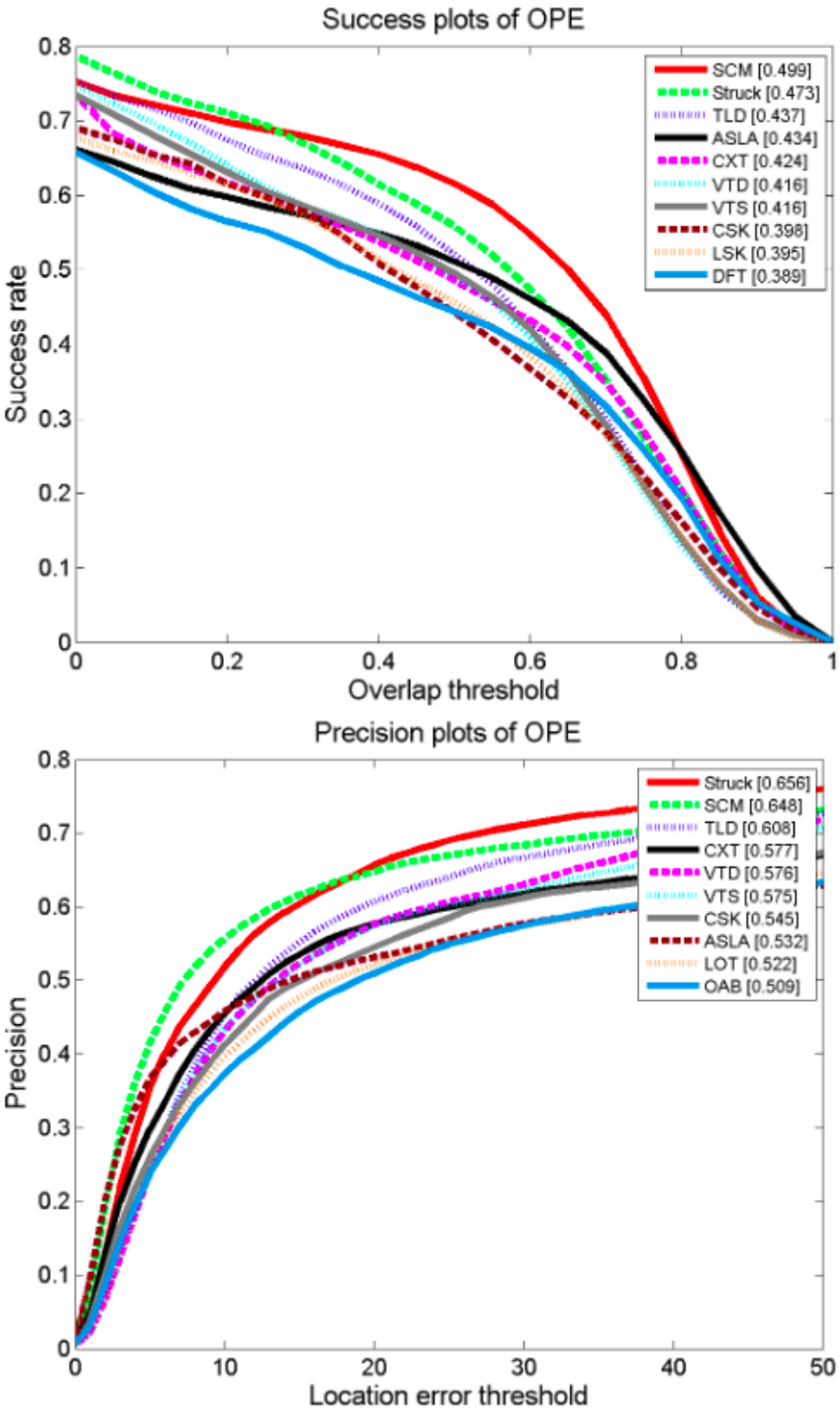


图2 Top 10 跟踪算法 OPE

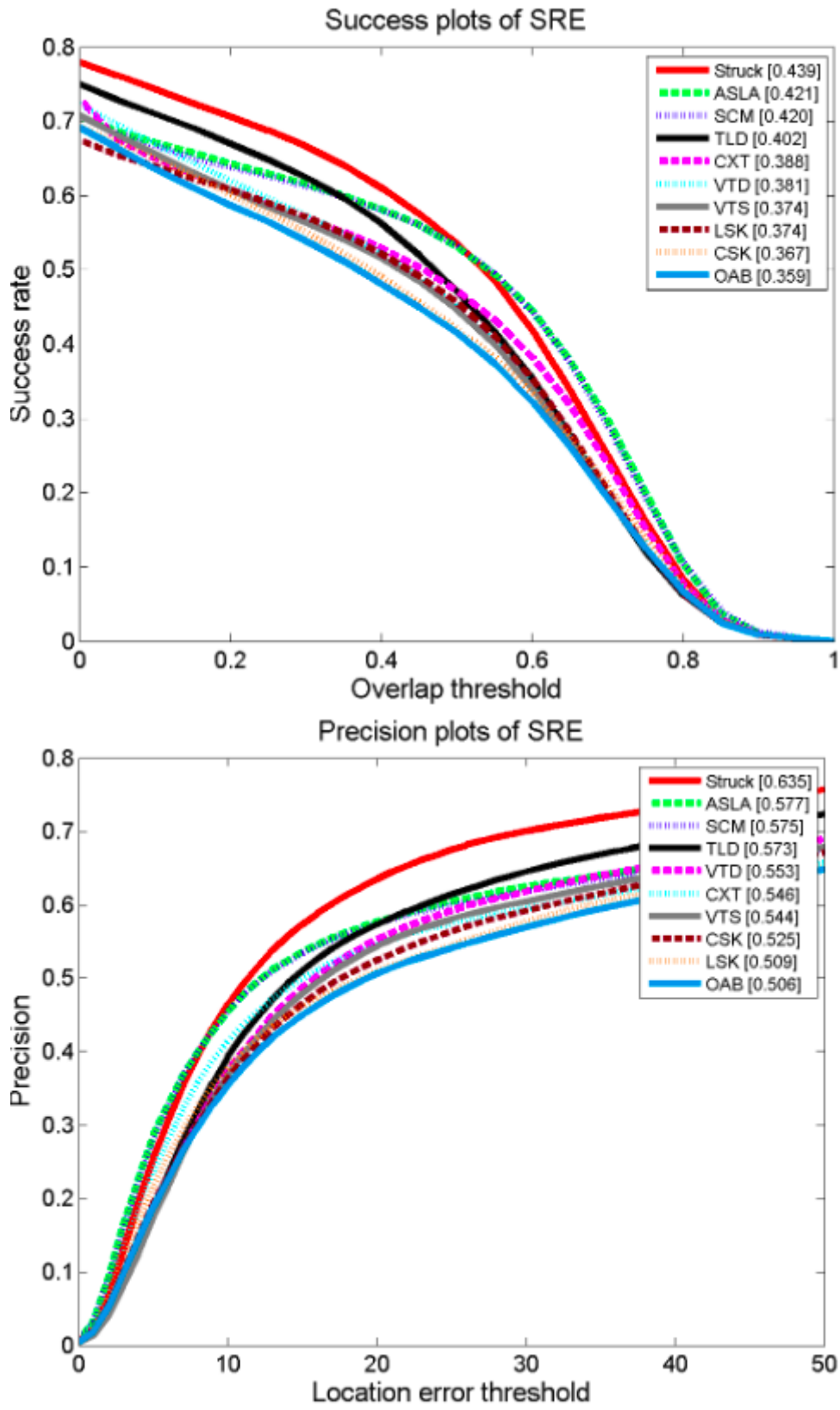


图3 Top10 跟踪算法 SRE

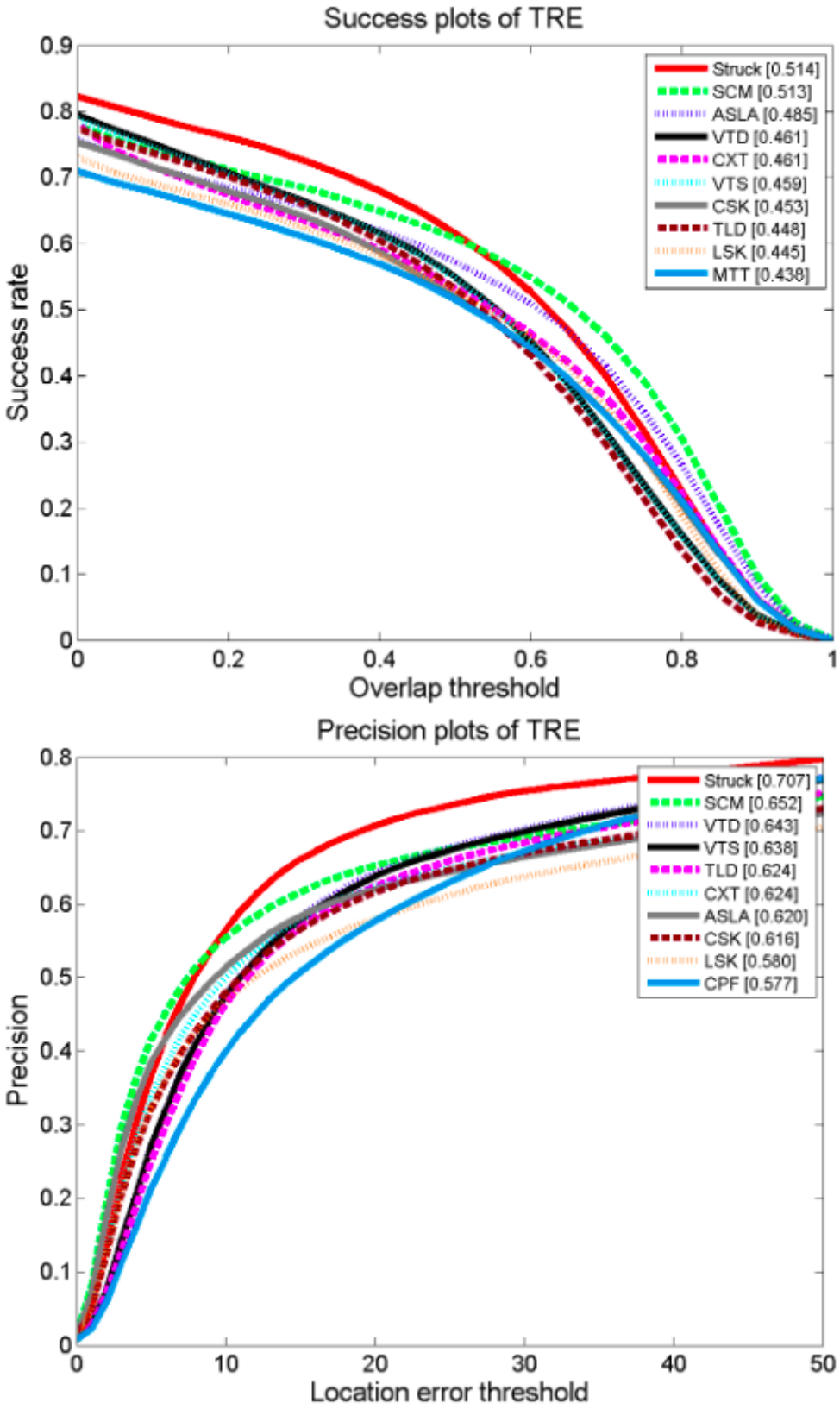


图4 Top10 跟踪算法 TRE # 7 航拍车辆跟踪算法

笔者简单分为基于传统机器学习和基于深度学习类的跟踪算法。标准为

- 目标检测：是否采用基于深度学习的算法，例如 R-CNN，Faster RCNN等；
- 跟踪算法：是否基于深度学习抽取运动特征。

7 航拍追车算法

7.1 基于传统算法

Blob Tracking 算法[14]: 无人机飞行时利用背景减除法检测出移动目标，然后根据 appearance 和 shape计算前后两帧相似度，超过一定阈值实时跟踪，准确率和速度文章没有谈到，**准确率和算力未知**。

Singular Value Decomposition 算法: 应用车辆行驶距离，图像颜色通道的归一化系数，车辆速度等相关信息进行跟踪，根据Navteq (www.navteq.com) 和 Atkis (www.atkis.de) 数据集测试，街道车辆49个，跟对26个，**准确率53%，目标丢失23个，帧率未知**。

ATLD: aerial tracking learning detection 算法，数据集来[17,18,19]，**文章未给出帧率，如果是标准的TLD，应该是28.1 FPS。准确率评估如下**

Sequence	Frames	TLD P/R/F	ATLD P/R/F
Biker[4]	1989	0.93/0.70/0.80	0.99/0.96/0.97
UAV1[4]	616	0.82/0.88/0.85	0.90/0.90/0.90
UAV 2[4]	1833	0.84/0.88/0.86	0.92/0.88/0.90
3 Car[2]	1301	0.83/0.84/0.84	0.90/0.89/0.89
Traffic 1[3]	156	**0.88/0.91/0.89**	0.88/0.91/0.89
Traffic 2[3]	227	0.82/0.80/0.81	0.85/0.90/0.87
Car[1]	945	0.91/0.95/0.94	0.99/0.99/0.99
Car Chase[1]	9928	0.87/0.77/0.78	0.96/0.98/0.97
Pedestrian 2[1]	338	0.89/0.90/0.89	0.93/0.95/0.94

图5 航拍和benchmark 数据集上对比ATLD 和 原始TLD Layer-based Tracker: 参考[20]，车和人的检测使用 HOG-based 分类器（KLT指的是 Kanade-Lucas-Tomasi based tracking），数据集采用VIVID2 UAV video sequence，架构如图6

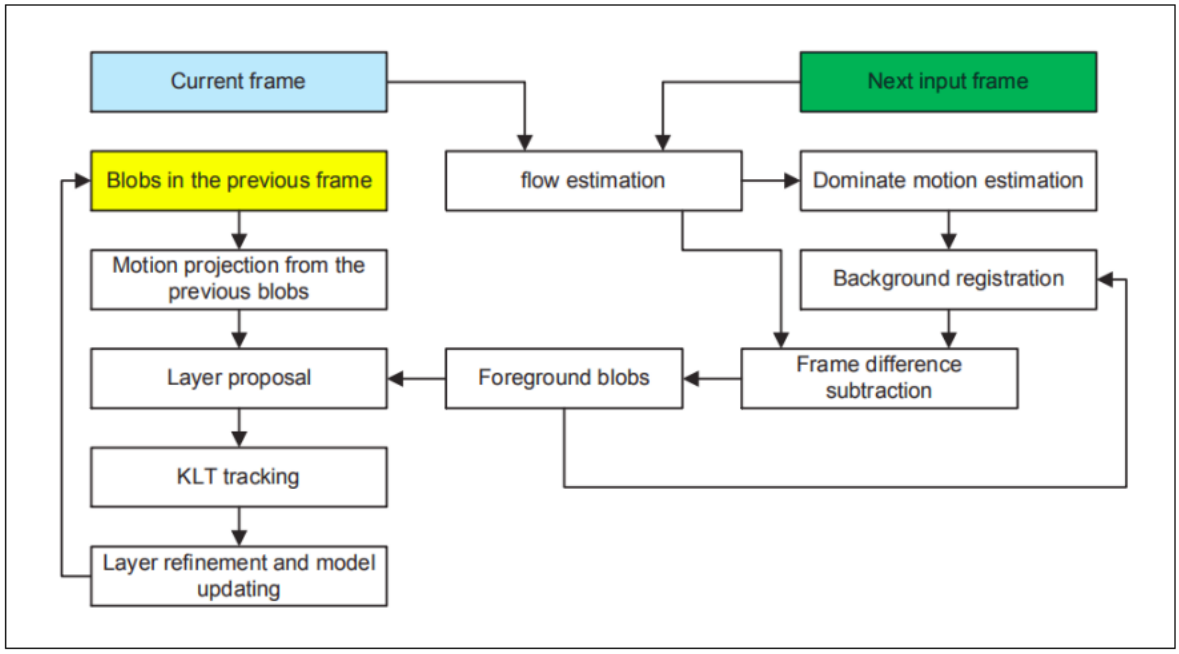


图6 - Layer-based Tracker架构

SNIFF-object Tracker: 参考[20]，同上采用VIVID2 UAV数据集，架构如图7

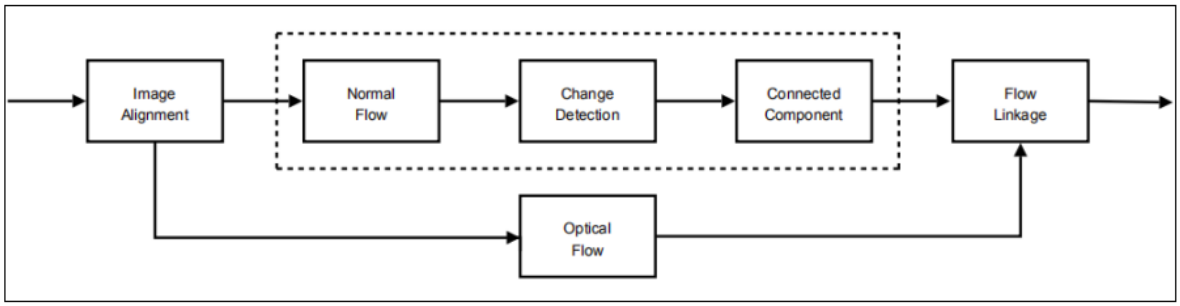


图7 SNIFF-object Tracker架构

使用上述系统化整合后的算法追车情况下：

表2 - 追车效果

Sequence ID	Detection accuracy	Miss detection per frame	False alarm per frame
V4V30002_046	0.000	0.120	0.000
V4V30002_047	0.753	0.724	0.000
V4V30002_048	0.876	0.331	0.005
V4V30002_049	0.929	0.078	0.000
V4V30002_050	0.947	0.107	0.100
V4V30003_011	0.966	0.006	0.067
V4V30003_014	0.622	2.350	0.179
V4V30003_017	0.969	0.039	0.005
V4V30004_005	0.789	0.202	0.157
V4V30003_015	0.868	0.095	0.286
V4V30004_020	0.879	0.107	0.339
V4V30004_021	0.889	0.084	0.241
V4V30004_024	0.923	0.04	0.408
V4V30004_028	0.897	0.022	0.186
V4V30004_029	0.894	0.404	0.061
V4V30004_043	0.945	0.297	0.073
V4V30004_044	0.653	2.293	0.005
V4V30004_046	0.826	0.655	0.293
V4V30004_049	0.960	0.005	0.000

追人效果

表3 追人效果

Sequence ID	Detection accuracy	Miss detection per frame	False alarm per frame
V4V30003_042	0.000	0.000	0.000
V4V30002_044	0.000	0.000	0.000
V4V30003_012	0.000	0.000	0.000
V4V30003_013	0.523	0.093	0.109
V4V30004_003	0.000	0.000	0.000
V4V30004_023	0.000	0.000	0.000
V4V30005_030	0.000	0.000	0.000
V4V30007_005	0.030	0.215	0.000
V4V30007_006	0.000	0.000	0.000
V4V30007_016	0.000	0.000	0.000
V4V30007_017	0.000	0.000	0.000
V4V30013_053	0.000	0.000	0.000

追人准确率低是因为人太小，宽大约2-5个像素，高大约5-10个像素，帧率未知。

光流法跟踪：参考[21]，特征点采集使用 SIFT算法， 边缘检测采用Prewitt算法，光流法采用标准Lucas-Kanade算法，架构如图8

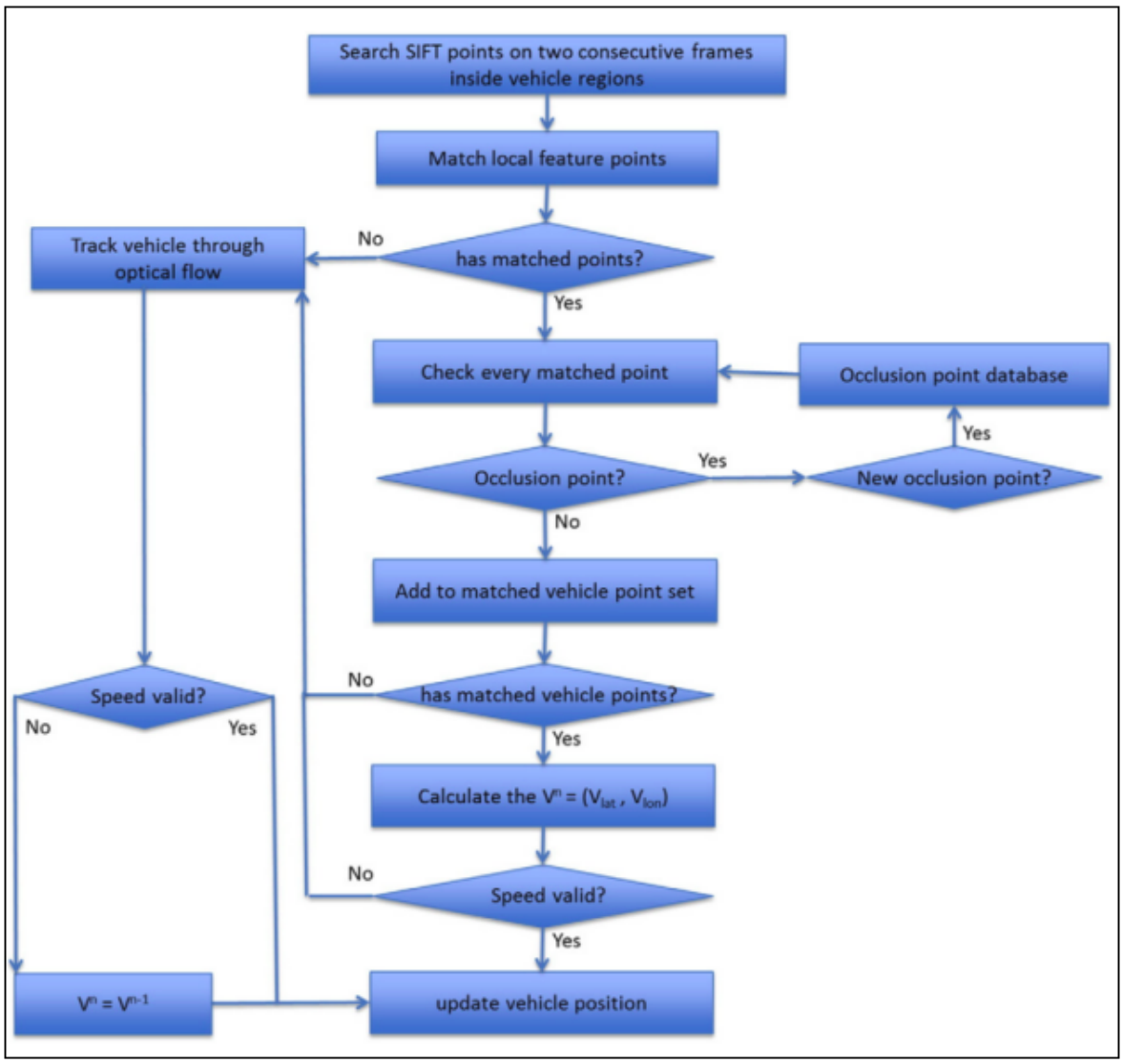


图8 光流法跟车

测试效果如下

表4 传统目标检测算法检测车辆

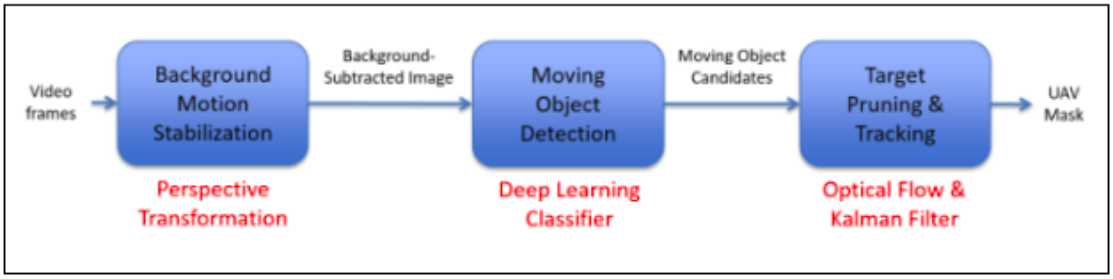
UAV altitude (m)/Road section length(m)	100/190	120/228	150/285
Vehicle average speed (m/s)	12.4	11.2	12.0
True positives	1999	1877	1520
False positives	0	0	0
False Negatives	2	3	61
Correctness	100%	100%	100%
Completeness	99.9%	99.8%	96.1%
Quality	99.9%	99.8%	96.1%

表5 跟踪效果

UAV altitude (m)/Road section length (m)	100/190	120/228	150/285
Number of vehicles	1999	1877	1520
Number of "lost vehicle"	0	11	32
Error rate	0%	0.6%	2.1%

卡尔曼滤波跟踪算法：

- 文献[22]：图片处理大小 820*432，高度92m，**准确率好，帧率未知**;
- 文献[25]：深度学习作为分类器，卡尔曼滤波作为跟踪算法，框架如下图



文献[25]**检测准确率**：

表5-A 深度学习分类器

	Only Motion Difference	Deep Learning with Appearance
Precision	0.630±0.11	0.819±0.09
Recall	0.766±0.15	0.798±0.10
F-Score	0.684±0.10	0.806±0.08

但跟踪效果未知。

CSRT Tracker：文献[23]采用Faster R-CNN目标检测算法，跟踪算法调研如下

- Boosting tracker：计算慢，跟踪效果不好；
- MIL：比Boosting tracker稍好些，但does a poor job reporting failure；
- KCF：比Boosting 和 MIL 计算快，但无法处理遮挡；
- MedianFlow：does a nice job of reporting failure但不能处理快速移动目标；
- TLD：有遮挡时表现的最好，但有着较高的 false-positive rate；
- MOSSE：计算很快但不准确。

最终，作者采用 CSRT Tracker[24]，架构如下

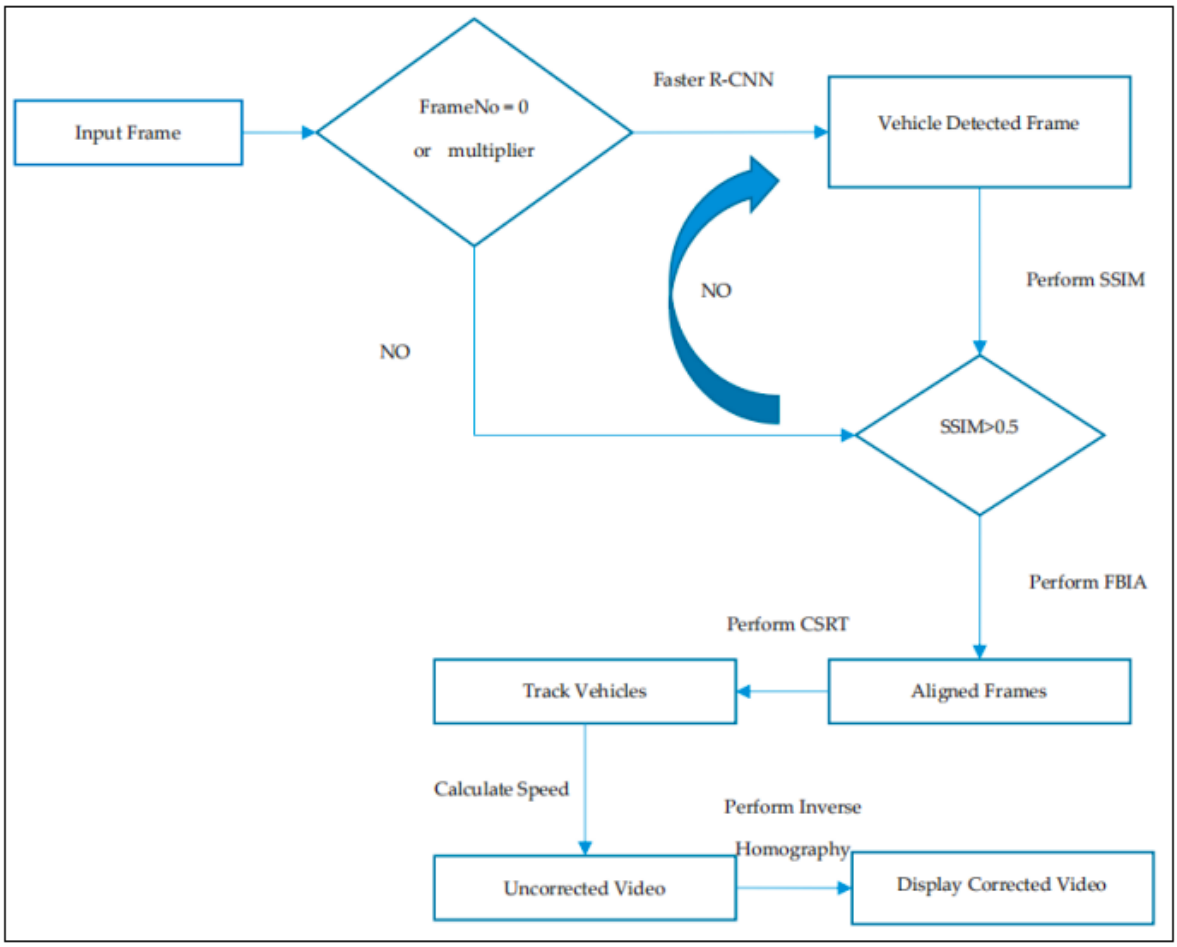


图9 车辆检测与跟踪算法流程图

效果如下：

表6 车辆检测效果

	Manual Count		Faster R-CNN Count		Accuracy	
	Small Vehicle	Large Vehicle	Small Vehicle	Large Vehicle	Small Vehicle	Large Vehicle
Glades Road/441 Intersection	506	2	437	2	86.37%	100%
Bluegrass	38	13	35	12	92.10%	92.30%
Burt Aaronson (Static)	19	0	19	0	100%	-
Burt Aaronson (moving)	39	0	33	0	84.61%	-
Total average					90.77%	96.15%

表7 车速检测效果

	Number of Cars Observed	Average Speed Manually Observed (mph)	Average Speed from the Framework (mph)	Accuracy %	RMSE (mph)
Glades Road/441 Intersection (Static drone)	43	0	0	100%	0
Glades Road/441 Intersection (Static drone)	10	30	30.5	98.33%	1.194
Glades Road/441 Intersection (Static drone)	15	25	26	96%	1.111
Bluegrass (Moving drone)	15	0	0	100%	0
Bluegrass (Moving drone)	5	12	11.5	95.83%	0.604
Burt Aaronson (Moving drone)	19	0	100%	100%	0
Burt Aaronson (Moving drone)	2	25	26	96%	1.044
Average				96.80%	0.564

7.2 基于深度学习

UAV Detection and Tracking Benchmark：文献[26]，MOT - Multi-Objects Tracking，方法有 CEM, CMOT, MDP等。注意，**表8和表9最后一列是GPU/CPU速度，其中加粗的是帧率超过 30 fps 的算法**。但该文章此处笔者认为书写有误，仅供参考。

表8 - Multi-Objects Tracking 效果

MOT methods	IDF	IDP	IDR	MOTA	MOTP	MT[%]	ML[%]	FP	FN	IDS	FM	Speed [fps]
Detection Input: Faster-RCNN [37]												
CEM [30]	10.2	19.4	7.0	−7.3	69.6	7.3	68.6	72,378	290,962	2,488	**4,248**	−/14.55
CMOT [2]	52.0	63.9	43.8	36.4	**74.5**	36.5	26.1	53,920	160,963	1,777	5,709	−/2.83
DSORT [48]	58.2	72.2	48.8	40.7	73.2	41.7	23.7	44,868	155,290	2,061	6,432	15.01/2.98
GOG [35]	0.4	0.5	0.3	34.4	72.2	35.5	25.3	41,126	168,194	14,301	12,516	−/436.52
IOUT [7]	23.7	30.3	19.5	36.6	72.1	37.4	25.0	42,245	163,881	9,938	10,463	−/**1438.34**
MDP [50]	**61.5**	**74.5**	**52.3**	**43.0**	73.5	**45.3**	**22.7**	46,151	**147,735**	**541**	4,299	−/0.68
SMOT [13]	45.0	55.7	37.8	33.9	72.2	36.7	25.7	57,112	166,528	1,752	9,577	−/115.27
SORT [6]	43.7	58.9	34.8	39.0	74.3	33.9	28.0	**33,037**	172,628	2,350	5,787	−/245.79
Detection Input: R-FCN [8]												
CEM [30]	10.3	18.4	7.2	−9.6	70.4	6.0	67.8	81,617	289,683	2,201	3,789	−/9.82
CMOT [2]	50.8	59.4	44.3	27.1	**78.5**	35.9	27.9	80,592	167,043	919	2,788	−/2.65
DSORT [48]	55.5	**67.3**	47.2	**30.9**	77.0	36.6	27.4	66,839	168,409	424	4,746	9.22/1.95
GOG [35]	0.3	0.4	0.3	28.5	77.1	34.4	28.6	60,511	176,256	6,935	6,823	−/433.94
IOUT [7]	44.0	47.5	40.9	26.9	75.9	**44.3**	**22.9**	98,789	**145,617**	4,903	6,129	−/**863.53**
MDP [50]	**55.8**	63.9	**49.5**	28.9	76.7	40.9	25.9	82,540	159,452	**411**	**2,705**	−/0.67
SMOT [13]	44.0	53.5	37.3	24.5	77.2	33.7	29.2	76,544	179,609	1,370	5,142	−/64.68
SORT [6]	42.6	58.7	33.5	30.2	**78.5**	29.5	31.9	**44,612**	190,999	2,248	4,378	−/209.31
Detection Input: SSD [27]												
CEM [30]	10.1	21.1	6.6	−6.8	70.4	6.6	74.4	64,373	298,090	1,530	**2,835**	−/11.62
CMOT [2]	49.4	53.4	46.0	27.2	75.1	38.3	23.5	98,915	146,418	2,920	6,914	−/0.90
DSORT [48]	51.4	**65.7**	42.2	33.6	**76.7**	27.9	26.9	**51,549**	173,639	**1,143**	8,655	15.00/3.46
GOG [35]	0.3	0.4	0.3	33.6	76.4	36.0	22.4	70,080	148,369	7,964	10,023	−/239.60
IOUT [7]	29.4	34.5	25.6	33.5	76.6	34.3	23.4	65,549	154,042	6,993	8,793	−/**976.47**
MDP [50]	**58.8**	63.2	**55.0**	**39.8**	76.5	**47.3**	**19.5**	79,760	**124,206**	1,310	4,539	−/0.13
SMOT [13]	41.9	45.9	38.6	27.2	76.5	34.9	22.9	95,737	149,777	2,738	9,605	−/11.59
SORT [6]	37.1	45.8	31.1	33.2	**76.7**	27.3	25.4	57,440	166,493	3,918	7,898	−/153.70
Detection Input: RON [25]												
CEM [30]	10.1	18.8	6.9	−9.7	68.8	6.9	72.6	78,265	293,576	2,086	**3,526**	−/9.98
CMOT [2]	57.5	65.7	51.1	36.9	**74.7**	**46.5**	**24.6**	69,109	**144,760**	1,111	3,656	−/0.94
DSORT [48]	58.3	67.9	51.2	35.8	71.5	43.4	25.7	67,090	151,007	698	4,311	17.45/4.02
GOG [35]	0.3	0.3	0.2	35.7	72.0	43.9	26.2	62,929	153,336	3,104	5,130	−/287.97
IOUT [7]	50.1	59.1	43.4	35.6	72.0	43.9	26.2	63,086	153,348	2,991	5,103	−/**1383.33**
MDP [50]	**59.9**	**69.0**	**52.9**	35.3	71.7	45.0	25.5	70,186	149,980	**414**	3,640	−/0.12
SMOT [13]	52.6	60.8	46.3	32.8	72.0	43.4	27.1	73,226	154,696	1,157	4,643	−/29.37
SORT [6]	54.6	66.9	46.1	**37.2**	72.2	40.8	28.0	**53,435**	159,347	1,369	3,661	−/230.55

表9 - Single Object Tracking 效果

SOT methods	BC	CR	OR	SO	IV	OB	SV	LO	Speed [fps]
MDNet [33]	**39.7/63.6**	**43.0/69.6**	**42.7/66.8**	44.4/78.4	**48.5**/76.4	**47.0**/72.4	**46.2/68.5**	**38.1/54.7**	0.89/0.28
ECO [9]	38.9/61.1	42.2/64.4	39.5/62.7	**46.1**/79.1	47.3/76.9	43.7/71.0	43.1/63.2	36.0/50.8	16.95/3.90
GOTURN [20]	38.9/61.1	42.2/64.4	39.5/62.7	**46.1**/79.1	47.3/76.9	43.7/71.0	43.7/63.2	36.0/50.8	**65.29**/11.70
SiamFC [5]	38.6/57.8	40.9/61.6	38.4/60.0	43.9/73.2	47.4/74.2	45.3/**73.8**	42.4/60.4	35.9/47.9	38.20/5.50
ADNet [52]	37.0/60.4	39.9/64.8	36.8/60.1	43.2/77.9	45.8/73.7	42.8/68.9	40.9/61.2	35.8/49.2	5.78/2.42
CFNet [43]	36.0/56.7	39.7/64.3	36.9/59.9	43.5/77.5	45.1/72.7	43.5/71.7	40.9/61.1	33.3/44.7	8.94/6.45
SRDCF [10]	35.3/58.2	39.0/64.2	36.5/60.0	42.2/76.4	45.1/74.7	41.7/70.6	40.2/59.6	32.7/46.0	−/14.25
SRDCFdecon [11]	36.0/57.4	39.0/61.0	36.6/57.8	43.1/73.8	45.5/72.3	42.9/69.5	38.0/54.9	31.5/42.5	−/7.26
C-COT [12]	34.0/55.7	39.0/62.3	34.1/56.1	44.2/79.2	41.6/72.0	37.2/66.2	37.9/55.9	33.5/46.0	0.87/0.79
MCPF [53]	31.0/51.2	36.3/59.2	33.0/55.3	39.7/74.5	42.2/73.1	42.0/73.0	35.9/55.1	30.1/42.5	1.84/0.89
CREST [41]	33.6/56.2	38.7/62.1	35.4/55.8	38.3/74.2	40.5/69.0	37.7/65.6	36.5/56.7	35.1/49.7	2.83/0.36
Staple-CA [32]	32.9/59.2	35.2/65.8	34.6/62.0	38.0/**79.6**	43.1/**77.2**	40.6/71.3	36.7/62.3	32.5/49.6	−/**42.53**
STCT [45]	33.3/56.0	36.0/61.3	34.3/57.5	38.3/71.0	40.8/69.9	37.0/63.3	37.3/59.9	31.7/46.6	1.76/0.09
PTAV [17]	31.2/57.2	35.2/63.9	30.9/56.4	38.0/79.1	38.1/69.6	36.7/66.2	33.3/56.5	32.9/50.3	12.77/0.10
CF2 [28]	29.2/48.6	34.1/56.9	29.7/48.2	35.6/69.5	38.7/67.9	35.8/65.1	29.0/45.3	28.3/38.1	8.07/1.99
HDT [36]	25.1/50.1	27.3/56.2	24.8/48.7	29.8/72.6	31.3/68.6	30.3/65.4	25.0/45.2	25.4/37.6	5.25/1.72
KCF [21]	23.5/45.8	26.7/53.4	24.4/45.4	25.1/58.1	31.1/65.7	29.7/65.2	25.4/49.0	22.8/34.4	−/39.26
SINT [42]	38.9/45.8	26.7/53.4	24.4/45.4	25.1/58.1	31.1/65.7	29.7/65.2	25.4/49.0	22.8/34.4	37.60/−
FCNT [44]	20.6/54.8	21.8/60.2	23.6/54.9	21.9/71.9	25.5/72.1	24.2/70.5	24.6/57.5	22.3/47.2	3.09/−

SOT - Single Object Tracking，该方法有 MDNet, SiamFC, KCF, SINT等。由于涉及方法较多，专业人士可参考[26]找到具体的框架和算法介绍。

最后附上github里基于深度学习的目标检测与跟踪论文、代码、数据集等链接：

- https://github.com/abhineet123/Deep-Learning-for-Tracking-and-Detection#paper_s_
- https://github.com/huanglianghua/mot-papers

- https://github.com/foolwood/benchmark_results

里面包含CVPR近5年目标检测和单、多目标跟踪的论文出处，代码出处，benchmark等。

8 参考文献

[1] http://www.doc88.com/p-9983830032983.html

[2]https://blog.csdn.net/u014203453/article/details/77598997

[3]https://www.cnblogs.com/youyou0/p/10250326.html

[4] http://islab.ulsan.ac.kr/files/announcement/513/rcnn_pami.pdf

[5] https://dl.acm.org/citation.cfm?id=2920125

[6] https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[7] https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

[8] https://www.cs.unc.edu/~wliu/papers/ssd.pdf

[9] https://ieeexplore.ieee.org/document/7852775

[10] https://ieeexplore.ieee.org/document/7122912

[11] http://www.wanfangdata.com.cn/details/detail.do?_type=perio&id=kxjsygc201532032

[12] http://cvlab.hanyang.ac.kr/~jwlim/files/cvpr13benchmark.pdf

[13] https://pdfs.semanticscholar.org/fdb3/9076ea464f9e50c36335fee3d08aa8a19cdf.pdf

[14] https://www.cs.cmu.edu/~saada/Publications/2006_SPIE_COCOA.pdf

[15] https://pdfs.semanticscholar.org/7dde/a8c30b46c98de7bc9a6f51b8fe33633f4ace.pdf?_ga=2.48718382.964610184.1567829537-1839287013.1567829537

[16] https://pdfs.semanticscholar.org/7dde/a8c30b46c98de7bc9a6f51b8fe33633f4ace.pdf?_ga=2.48718382.964610184.1567829537-1839287013.1567829537

[17] http://crcv.ucf.edu/data/UCF_Aerial_Action.php

[18] http://i21www.ira.uka.de/image_sequences/

[19] http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/dataset

[20] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.729&rep=rep1&type=pdf

[21] https://www.researchgate.net/publication/303536189_Detecting_and_tracking_vehicles_in_traffic_by_unmanned_aerial_vehicles

[22] https://e-sciencecentral.org/upload/ijfis/pdf/ijfis-18-182.pdf

[23] Debojit Biswas, Hongbo Su, Chengyi Wang, and Aleksandar Stevanovic. Speed Estimation of Multiple Moving Objects from a Moving UAV Platform, International Journal of Geo-Information, 2019

[24] https://pdfs.semanticscholar.org/b16a/583ee173f222c690242aaff7925838893fe8.pdf

[25] https://pdfs.semanticscholar.org/effb/52faedbfb92e2aac624fdd46a641b20fbe51.pdf

[26] http://openaccess.thecvf.com/content_ECCV_2018/papers/Dawei_Du_The_Unmanned_Aerial_ECCV_2018_paper.pdf