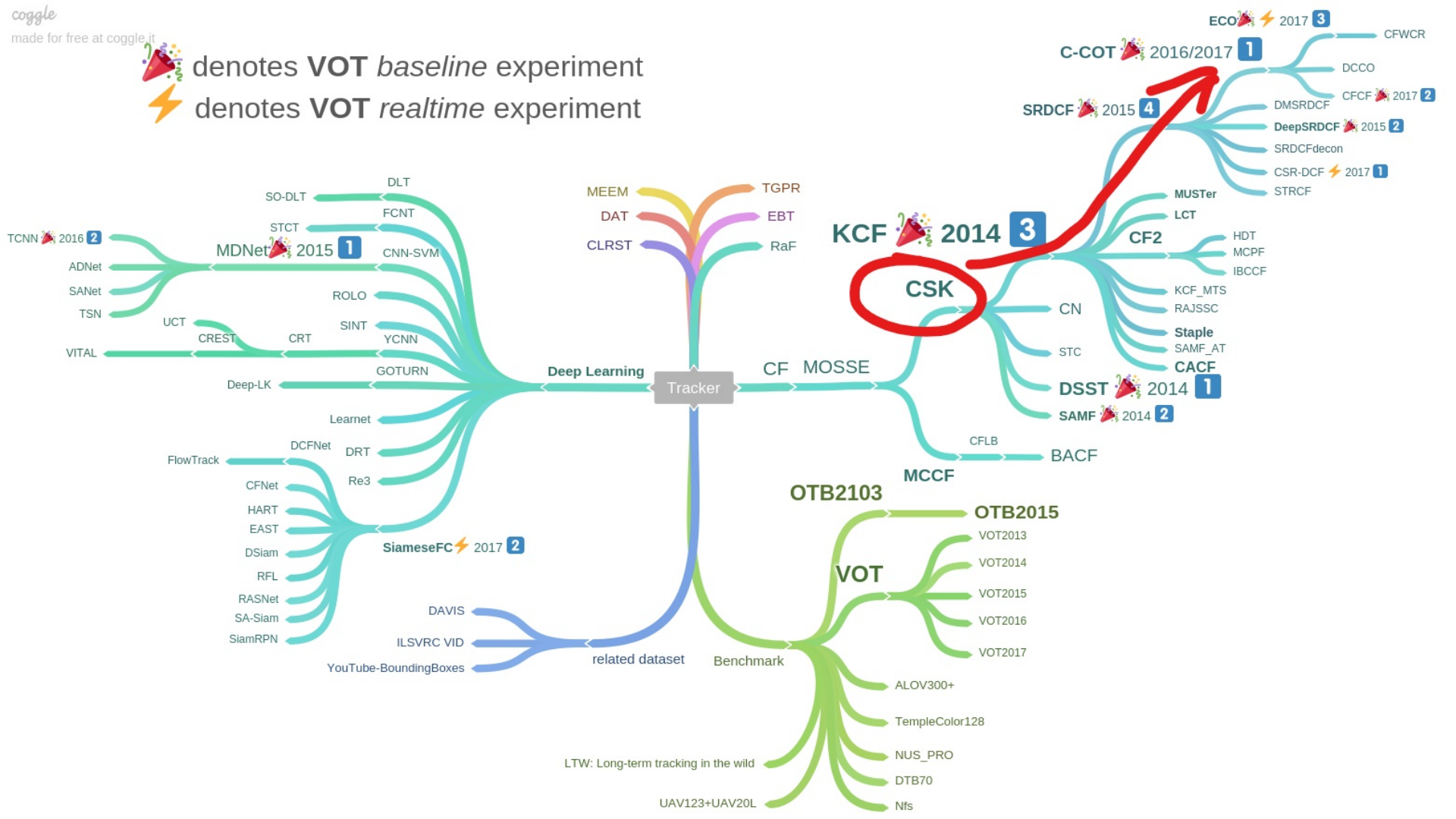
CSK跟踪（利用循环结构和核函数进行检测跟踪）

参考文献：

Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista “Exploiting the Circulant Structure of Tracking-by-detection with Kernels”, ECCV, 2012

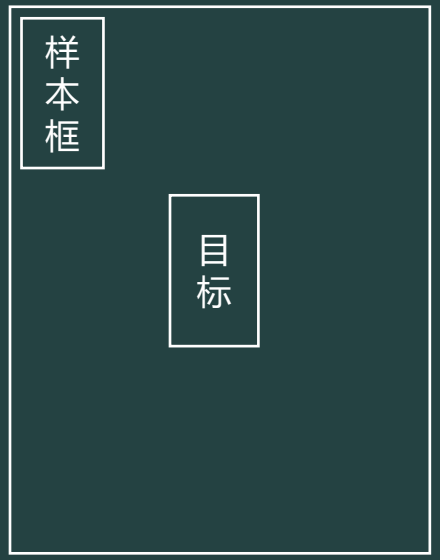


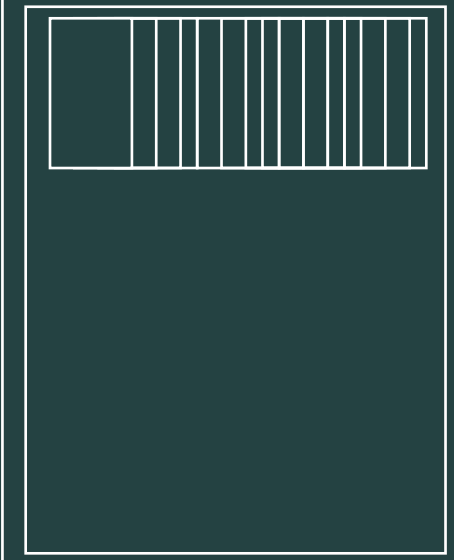
前一帧X：

训练一个分类器

后一帧Z：

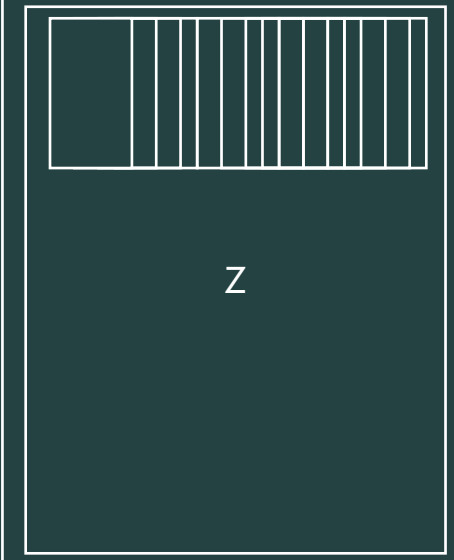
稠密采样：



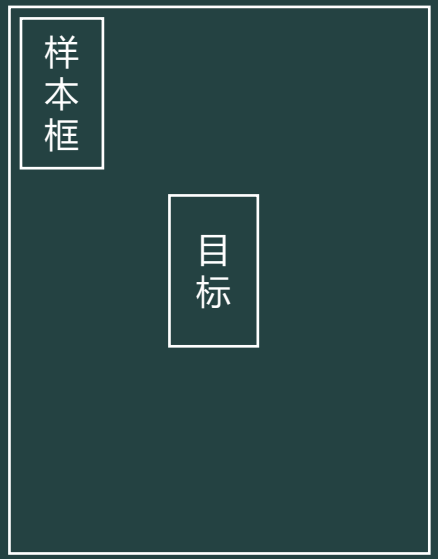


获得训练集x_i

稠密采样：



获得下一帧的测试集z_i



训练样本集(x_i, y_i)

Y是该样本的标签，距离目标越远越负，离目标越近越正

最小化问题转化为：$\min(y - f(x))^2 + \lambda||w||^2$

低维$x_i \rightarrow \varphi(x_i)$高维　　$z_i \rightarrow \varphi(z_i)$

在训练集中训练出一个权值w，**在下一帧中对z进行分类：**

$$f(z_i) = wz_i$$

w应该为$\varphi(X) = [\varphi(1), \varphi(2), \varphi(3) \ldots \varphi(i)]$空间中张成的一个向量

$$w = \Sigma_i \alpha_i \varphi(x_i)$$

$$f(z) = wz = \partial\varphi(x)\varphi(z)$$

求解只用到内积运算，而在低维输入空间又存在某个函数K(x, x′)，它恰好等于在高维空间中这个内积，

$$\varphi(x)\varphi(z) = K(x, z)$$

$$f(z) = \partial K(x, z) = \partial K$$

$$\min(y - f(x))^2 + \lambda||w||^2 \qquad |w||^2 = (\partial\varphi(x))^T(\partial\varphi(x)) = \partial^T K\partial$$

$$\Downarrow$$

$$\min(Y - K\partial)^2 + \lambda\partial^T K\partial$$

$$\Downarrow$$

$$(Y - K\partial)^2(Y - K\partial) + \lambda\partial^T k\partial = 0$$

求解得：$\partial = (K - \lambda I)^{-1}Y$

循环矩阵

$$C(u) = \begin{pmatrix} u_1 & u_2 & u_2 & \dots & u_2 \\ u_n & u_1 & u_2 & \dots & u_{n-1} \\ u_{n-1} & u_n & u_{n-1} & \dots & u_{n-2} \\ u_{n-2} & u_{n-1} & u_n & \dots & u_{n-3} \end{pmatrix}$$

$$u = \begin{pmatrix} u_1 & u_2 & u_3 \dots & u_n \end{pmatrix}$$

$$C(u)x = b$$
$$u * x = b$$
$$F(u)F(x) = F(b)$$
$$x = F^{-1}\left(\frac{F(b)}{F(u)}\right)$$

$$\partial = (K - \lambda I)^{-1}Y$$
$$(K - \lambda I)\partial = Y$$
$$C(k - \lambda(\delta))\partial = Y$$
$$\partial = F^{-1}\left(\frac{F(Y)}{F(k) - \lambda}\right)$$

高斯核函

Figure 1

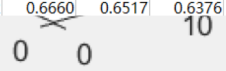
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)

变量 - k

k

120x48 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	0.9969	0.9887	0.9761	0.9595	0.9396	0.9171	0.8925	0.8665	0.8395	0.8123	0.7852	0.7587	0.7331	0.7089	0.6862	0.6653	0.6464	0.6298	0.6154	0.6035	0.5942	0.5875	0.5835
2	0.9993	0.9961	0.9879	0.9753	0.9587	0.9388	0.9163	0.8918	0.8657	0.8388	0.8116	0.7846	0.7581	0.7326	0.7084	0.6858	0.6650	0.6462	0.6295	0.6152	0.6033	0.5940	0.5874	0.5834
3	0.9973	0.9941	0.9859	0.9734	0.9569	0.9371	0.9147	0.8903	0.8644	0.8376	0.8105	0.7835	0.7571	0.7317	0.7076	0.6851	0.6644	0.6456	0.6291	0.6148	0.6030	0.5937	0.5871	0.5832
4	0.9943	0.9911	0.9829	0.9705	0.9542	0.9346	0.9124	0.8882	0.8624	0.8358	0.8088	0.7820	0.7558	0.7305	0.7066	0.6841	0.6635	0.6449	0.6284	0.6142	0.6025	0.5933	0.5867	0.5828
5	0.9905	0.9873	0.9791	0.9669	0.9508	0.9314	0.9094	0.8854	0.8599	0.8335	0.8068	0.7802	0.7541	0.7290	0.7052	0.6829	0.6624	0.6439	0.6275	0.6135	0.6018	0.5927	0.5862	0.5823
6	0.9861	0.9827	0.9747	0.9626	0.9467	0.9275	0.9059	0.8821	0.8569	0.8307	0.8043	0.7779	0.7521	0.7272	0.7035	0.6814	0.6611	0.6427	0.6265	0.6125	0.6010	0.5920	0.5855	0.5817
7	0.9811	0.9777	0.9697	0.9577	0.9420	0.9231	0.9018	0.8783	0.8534	0.8275	0.8014	0.7753	0.7497	0.7250	0.7016	0.6797	0.6595	0.6413	0.6253	0.6115	0.6000	0.5911	0.5848	0.5810
8	0.9756	0.9721	0.9641	0.9523	0.9368	0.9183	0.8972	0.8741	0.8495	0.8239	0.7981	0.7722	0.7469	0.7225	0.6994	0.6777	0.6578	0.6398	0.6239	0.6102	0.5989	0.5901	0.5839	0.5802
9	0.9697	0.9661	0.9582	0.9465	0.9313	0.9130	0.8922	0.8694	0.8452	0.8200	0.7944	0.7689	0.7439	0.7198	0.6969	0.6755	0.6558	0.6380	0.6223	0.6088	0.5977	0.5890	0.5829	0.5793
10	0.9632	0.9597	0.9518	0.9403	0.9253	0.9072	0.8868	0.8644	0.8404	0.8156	0.7904	0.7653	0.7406	0.7168	0.6942	0.6730	0.6536	0.6360	0.6206	0.6073	0.5963	0.5878	0.5817	0.5782
11	0.9563	0.9528	0.9450	0.9336	0.9188	0.9011	0.8810	0.8589	0.8354	0.8109	0.7861	0.7613	0.7370	0.7135	0.6912	0.6704	0.6512	0.6339	0.6187	0.6056	0.5948	0.5864	0.5805	0.5770
12	0.9488	0.9453	0.9377	0.9265	0.9120	0.8946	0.8748	0.8531	0.8300	0.8059	0.7815	0.7571	0.7331	0.7100	0.6881	0.6676	0.6487	0.6317	0.6167	0.6038	0.5932	0.5850	0.5792	0.5758
13	0.9408	0.9374	0.9299	0.9190	0.9047	0.8876	0.8683	0.8470	0.8242	0.8006	0.7765	0.7526	0.7290	0.7063	0.6847	0.6646	0.6460	0.6293	0.6145	0.6019	0.5915	0.5834	0.5777	0.5744
14	0.9323	0.9290	0.9217	0.9110	0.8970	0.8803	0.8613	0.8404	0.8181	0.7949	0.7714	0.7478	0.7247	0.7024	0.6812	0.6614	0.6432	0.6268	0.6123	0.5999	0.5897	0.5818	0.5762	0.5730
15	0.9234	0.9201	0.9130	0.9026	0.8890	0.8726	0.8540	0.8336	0.8117	0.7890	0.7659	0.7428	0.7202	0.6983	0.6775	0.6581	0.6403	0.6242	0.6100	0.5978	0.5878	0.5801	0.5746	0.5715
16	0.9141	0.9109	0.9040	0.8938	0.8805	0.8645	0.8464	0.8264	0.8050	0.7828	0.7602	0.7376	0.7155	0.6940	0.6737	0.6547	0.6372	0.6215	0.6076	0.5957	0.5859	0.5783	0.5730	0.5699
17	0.9044	0.9013	0.8946	0.8847	0.8717	0.8561	0.8384	0.8189	0.7980	0.7763	0.7542	0.7322	0.7105	0.6896	0.6697	0.6511	0.6341	0.6187	0.6051	0.5934	0.5839	0.5764	0.5712	0.5683
18	0.8944	0.8915	0.8849	0.8753	0.8626	0.8474	0.8301	0.8111	0.7907	0.7696	0.7481	0.7266	0.7054	0.6850	0.6656	0.6475	0.6308	0.6158	0.6025	0.5911	0.5818	0.5745	0.5694	0.5665
19	0.8842	0.8813	0.8749	0.8655	0.8532	0.8384	0.8216	0.8030	0.7832	0.7626	0.7417	0.7207	0.7002	0.6803	0.6614	0.6437	0.6274	0.6128	0.5998	0.5887	0.5796	0.5725	0.5675	0.5646
20	0.8737	0.8709	0.8646	0.8555	0.8435	0.8291	0.8128	0.7948	0.7755	0.7555	0.7351	0.7148	0.6948	0.6754	0.6570	0.6398	0.6240	0.6097	0.5971	0.5862	0.5773	0.5704	0.5655	0.5627
21	0.8631	0.8602	0.8541	0.8453	0.8336	0.8197	0.8038	0.7863	0.7676	0.7482	0.7284	0.7087	0.6892	0.6705	0.6526	0.6359	0.6205	0.6065	0.5942	0.5837	0.5749	0.5682	0.5634	0.5607
22	0.8522	0.8494	0.8435	0.8348	0.8236	0.8100	0.7946	0.7776	0.7595	0.7407	0.7216	0.7024	0.6836	0.6654	0.6481	0.6318	0.6168	0.6033	0.5913	0.5810	0.5725	0.5659	0.5612	0.5586
23	0.8412	0.8384	0.8326	0.8243	0.8133	0.8002	0.7853	0.7689	0.7514	0.7332	0.7146	0.6961	0.6779	0.6602	0.6434	0.6277	0.6132	0.6000	0.5883	0.5783	0.5700	0.5635	0.5590	0.5564
24	0.8300	0.8274	0.8217	0.8136	0.8030	0.7903	0.7759	0.7601	0.7431	0.7256	0.7077	0.6898	0.6721	0.6551	0.6388	0.6235	0.6094	0.5966	0.5853	0.5755	0.5674	0.5611	0.5567	0.5542
25	0.8188	0.8162	0.8108	0.8029	0.7926	0.7804	0.7665	0.7512	0.7349	0.7179	0.7007	0.6834	0.6663	0.6498	0.6341	0.6193	0.6056	0.5932	0.5822	0.5727	0.5648	0.5587	0.5544	0.5520
26	0.8076	0.8051	0.7998	0.7922	0.7823	0.7705	0.7571	0.7424	0.7267	0.7103	0.6937	0.6770	0.6606	0.6446	0.6294	0.6151	0.6018	0.5898	0.5791	0.5698	0.5622	0.5562	0.5521	0.5497
27	0.7964	0.7939	0	0.7815	0.7719	0.7606	0.7477	0.7335	0.7184	0.7027	0.6867	0.6706	0.6548	0.6394	0.6247	0.6108	0.5980	0.5863	0.5759	0.5669	0.5595	0.5537	0.5497	0.5474
28	0.7852	0.7828	0.7779	0.7708	0.7617	0.7507	0.7384	0.7248	0.7103	0.6952	0.6798	0.6643	0.6490	0.6342	0.6199	0.6065	0.5941	0.5828	0.5727	0.5640	0.5568	0.5512	0.5473	0.5451
29	0.7740	0.7717	0.7670	0.7602	0.7515	0.7410	0.7291	0.7161	0.7022	0.6877	0.6729	0.6580	0.6433	0.6290	0.6152	0.6023	0.5902	0.5793	0.5695	0.5611	0.5541	0.5487	0.5449	0.5428
30	0.7630	0.7608	0.7563	0.7498	0.7414	0.7313	0.7199	0.7075	0.6941	0.6802	0.6660	0.6517	0.6376	0.6238	0.6105	0.5980	0.5863	0.5757	0.5663	0.5582	0.5514	0.5462	0.5425	0.5405

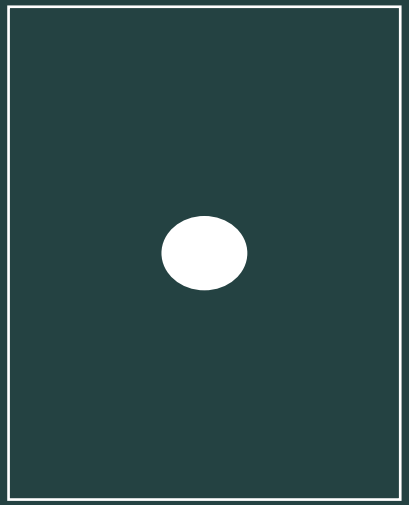


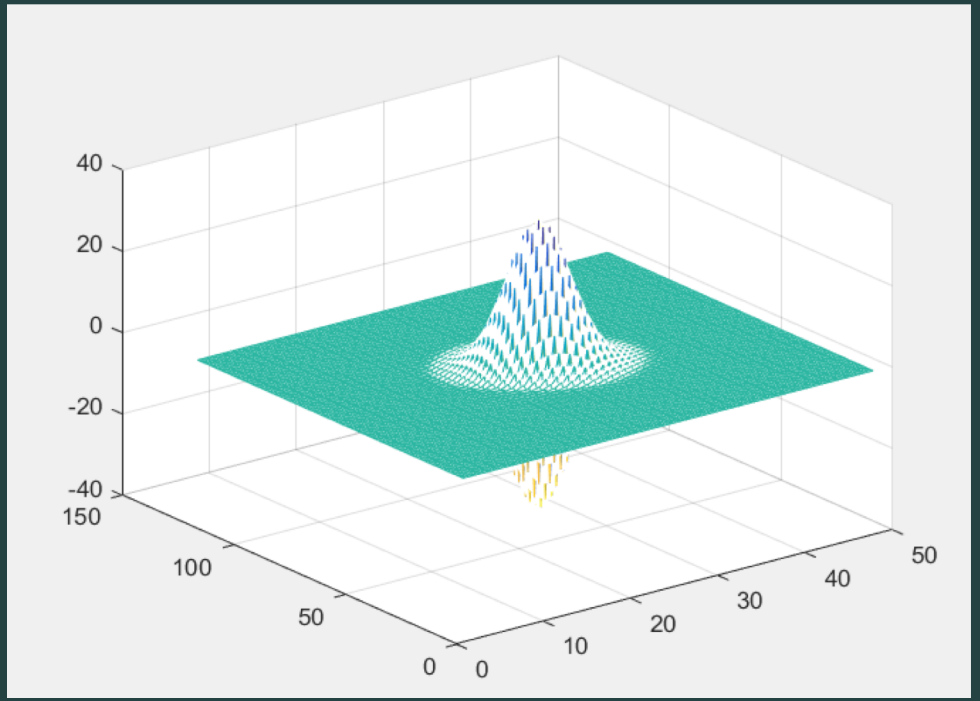
$$Y' = \partial K$$

最大响应的位置就为下一帧中最大可能出现的位置

边界问题

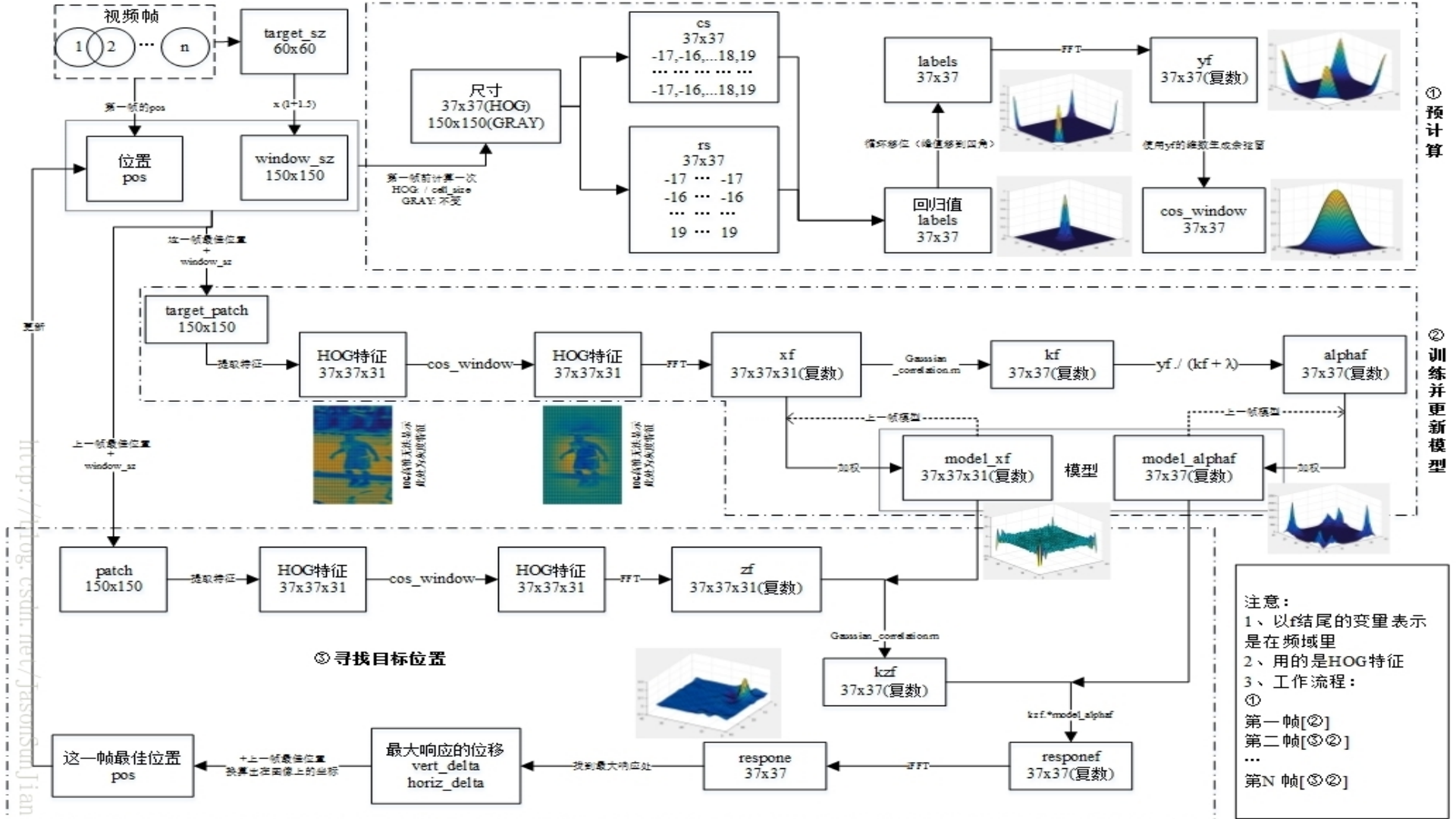
训练集标签Y的设定



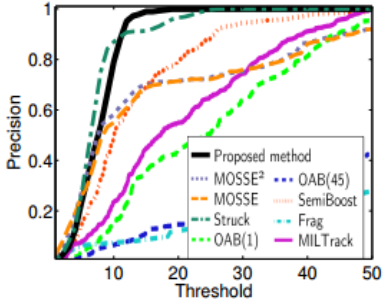


Algorithm 1 : MATLAB code for our tracker, using a Gaussian kernel
It is possible to reuse some values, reducing the number of FFT calls. An implementation with GUI is available at: `http://www.isr.uc.pt/~henriques/`

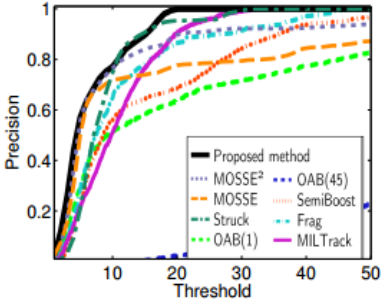
```
% Training image x (current frame) and test image z (next frame)
% must be pre-processed with a cosine window. y has a Gaussian
% shape centered on the target. x, y and z are M-by-N matrices.
% All FFT operations are standard in MATLAB.

function alphaf = training(x, y, sigma, lambda)        % Eq. 7
  k = dgk(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detection(alphaf, x, z, sigma)   % Eq. 9
  k = dgk(x, z, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = dgk(x1, x2, sigma)                        % Eq. 16
  c = fftshift(ifft2(fft2(x1) .* conj(fft2(x2))));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2*c;
  k = exp(-1 / sigma^2 * abs(d) / numel(x1));
end
```


视频帧
1
2
n
target_sz
60x60
第一帧的pos
x (1+1.5)
位置
pos
window_sz
150x150
尺寸
37x37(HOG)
150x150(GRAY)
第一帧前计算一次
HOG：cell_size
GRAY：不变
cs
37x37
-17,-16,...18,19
-17,-16,...18,19
rs
37x37
-17 -16 19
-16 -16 19
19 19
labels
37x37
FFT
yf
37x37(复数)
循环移位（峰值移到四角）
使用yf的维数生成余弦窗
回归值
labels
37x37
cos_window
37x37
①预计算
上一帧最佳位置
+
window_sz
target_patch
150x150
提取特征
HOG特征
37x37x31
cos_window
HOG特征
37x37x31
FFT
xf
37x37x31(复数)
Gaussian
_correlation.m
kf
37x37(复数)
yf ./ (kf + λ)
alphaf
37x37(复数)
上一帧模型
加权
model_xf
37x37x31(复数)
模型
model_alphaf
37x37(复数)
更新
②训练并更新模型
patch
150x150
提取特征
HOG特征
37x37x31
cos_window
HOG特征
37x37x31
FFT
zf
37x37x31(复数)
Gaussian_correlation.m
kzf
37x37(复数)
③寻找目标位置
kzf.*model_alphaf
responef
37x37(复数)
IFFT
respone
37x37
找到最大响应处
最大响应的位移
vert_delta
horiz_delta
+上一帧最佳位置
换算出在图像上的坐标
这一帧最佳位置
pos
注意：
1、以f结尾的变量表示是在频域里
2、用的是HOG特征
3、工作流程：
①
第一帧[②]
第二帧[③②]
…
第N 帧[③②]

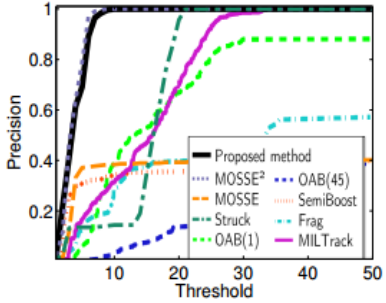
在六组序列上的实验结果



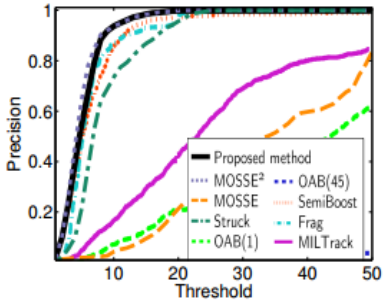
(a) coke11



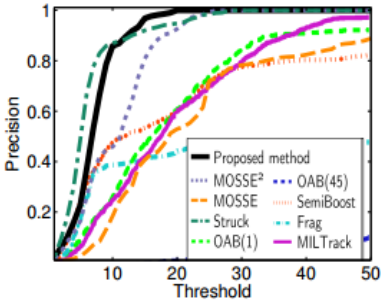
(b) sylvester



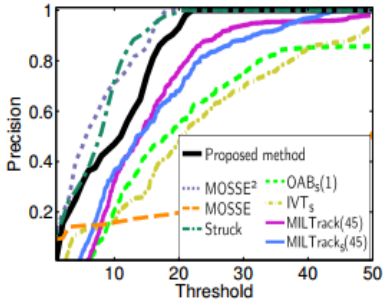
(c) dollar



(d) faceocc



(e) faceocc2



(f) twinings

结果：精度更好，速度也是在高速上百FPS。

精度更好：这是因为在下一帧中进行了稠密取样，测试了下一帧中每一个位置目标出现的概率。
而速度也能保持很快的原因就在于利用了核函数使得这么多的样本构建成为循环矩阵结构，从而使得所有的运算在傅里叶频域中运算，利用matlab自带函数fft2，使得速度达到高速状态。