Multiple Channels Distribution Fields tracker

Aurélien Greffard

University of Ljubljana,
Faculty of Computer and Information Science
Advanced topics in Computer Vision
June 2014

Abstract. This paper relates my research on possible ways to extend the Distribution Fields tracker initially proposed by Laura-Sevilla-Lara and Eril Learned-Miller in 2012. An improvement of this tracker was already submitted in 2013 by Michael Felsberg using Channel-representations as an approximation of kernel density estimates. My objective was to explore other ways of improvements: how to efficiently improve the accuracy and robustness of the DF tracker? We will review here the implementation and results of proposed improvements, based on the VOT2013 Challenge.

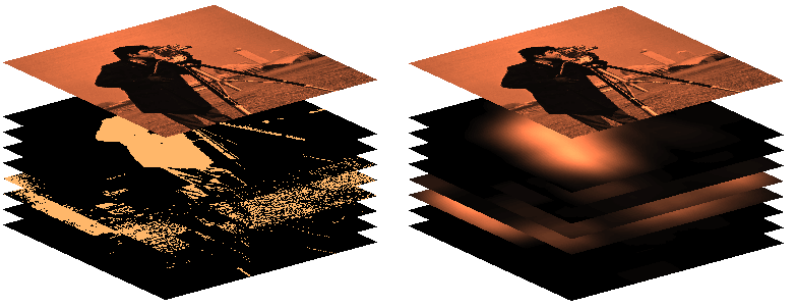
Keywords: Distribution Fields, Tracking, Gradient descent, Color information, Scale adaptation, Kalman filter, VOT2013

1 Introduction

My starting point was the Distribution Fields tracker, available at `http://people.cs.umass.edu/~lsevilla/trackingDF.html`. I refer the reader to the original paper for more details about theory and implementation.

1.1 Distribution Fields for tracking

This tracker is based on Distribution Fields (DF) as a visual model for the target. The principle is to explode the image into different layers, corresponding to similar intensity pixels, and to smooth these layers separately to be able to apply gradient descent to reach a local minimum and match the target.



Explosion and smoothness of the original image

The main advantage is that we can smooth the image to perform gradient descent without losing information: the DFs contain the same information as the original representation, but in a larger representation.

1.2 Implementation details

The first step is the explosion of the image in different layers representing different ranges of values (similar intensity pixels). All the pixels from the original image with an intensity falling into the value set of a certain layer are marked 1 in this layer, the other ones are marked 0:

$$\mathrm{d(i,\ j,\ k)} = \begin{cases} 1, & \text{if I(i, j) == k} \\ 0, & \text{otherwise} \end{cases}$$

Then each layer is smoothed with a gaussian function (and becomes then a probability density function). The smooth is performed in three dimensions: both directions of each layer, and across the image layers.

Finally a gradient descent is performed, using L1 distance to test the matching between a possible target and the visual model.

At the end, the target model is updated:

$\mathrm{d}_{model} = 0.95 * d_{model} + 0.5 * d_{target}$

2 Methods

2.1 Multiple channels information

The initial DF tracker is based on grayscale images, so it doesnt use all information of the image, in the case of multiple channels images like RGB sequences. The idea is to use all channels to have a more performant visual model, and then a more accurate tracker. Therefore we add a new dimension to the DF, equal to the number of channels. We obtain one DF for each channel, lets call this new model *aMultiple Channels Distribution Fields* (MCDF).

Our model contains now more informations than previously. The most important thing is now to find the best way to use this MCDF for tracking. The naive way is simply to extend the comparison between models by taking all the channels into account in the same measure, and to perform the gradient descent according to this single measure.

A better idea is to perform the tracking using each channel separately: the gradient descent is executed several times, comparing only one channel at a time. Thereby, for a RGB sequence, we obtain three possible locations of the target. We have now several possibilities for choosing the definitive location:

- A simple averaging between the different positions
- Keep only the position obtained with the channel with lowest distance to the model
- A weighted averaging: the positions obtained with the channels with the lowest distance to the model have a higher weight

After different experiments, the most accurate solution was the first one (although this is not the most intuitive): a simple averaging between all different locations. The results are really positive.

2.2 Scale adaptation

The other way to improve this tracker was to make it scale-adaptive. The idea, already used in other trackers, is to launch the tracker several times with different scales, to keep the best one and update the window size of the target and the size of the model.

Experiments revealed that the best solution was to launch the tracker three times, checking variations of 5% of the previous window size, and seemed to be really efficient :

Scale adaptation on the sequence juice

However, this solution requires also much more time, and it was not conceivable to run the VOT experiments on this too slow tracker.

Then, another idea is to check different scales at the end of the process, without running the whole tracking several times. If we consider that our tracker is efficient and accurate enough to find the exact location of the target, then we can say that we already found the desired location, and we only need to adapt the size of the window. This solution was of course faster, but didnt provide expected results, as discussed in the next section. Indeed, the assumption of having a 100% accurate tracker is impossible, and checking the size from the found location can introduce even more inexactness in some cases, in some sequences.

2.3 Initial estimate of target

Finally, I tried to find a way to make a better prediction of the target before performing the gradient descent. The objectives are to reduce the execution time (the target is found in a shorter number of iterations) and to avoid the target losses (by avoiding to be stuck in a local optimum).

The solution implemented was to use a Kalman filter along the sequence, with a Nearly Constant Velocity model. The initial parameters used were:

System matrix: $A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Observation matrix: $C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

System noise covariance matrix: Q = eyes(size(A, 1));

Observation noise covariance matrix: R = 5*eyes(size(C, 1));

With an initial state: $S = \begin{pmatrix} x_{position} & y_{position} & x_{velocity} & y_{velocity} \\ x_{init} & y_{init} & 0 & 0 \end{pmatrix}^\top$

2.4 Other experiments

I also tried to explore different possible improvements by changing the parameters (number of bins, amount of scale) or using other distance measures (L2 distance, Bhattacharyya distance...). Unfortunately, and without surprise, none of these modifications were concluding: Sevilla-Lara and Learned-Miller took care about these details and kept the best parameters.

3 Results

We will review here the results obtained after each step of improvement. Launching the whole VOT experiment for each new implemented feature was not possible for time reasons, but it was possible to run it on the baseline experiment. In the following paragraphs we will then compare the results after each modification, with the initial DF tracker:

Table 1. Experiment *baseline* results for tracker *DF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.47	1.00	7.34
bolt	0.76	8.00	5.13
car	0.46	1.00	5.73
cup	0.78	0.00	2.54
david	0.64	0.00	3.01
diving	0.39	5.00	4.49
face	0.88	0.00	2.59
gymnastics	0.49	2.00	3.84
hand	0.49	2.00	4.10
iceskater	0.34	1.00	3.08
juice	0.49	0.00	4.13
jump	0.61	0.00	3.85
singer	0.37	0.00	1.91
sunshade	0.67	3.00	3.37
torus	0.63	1.00	4.27
woman	0.59	1.00	3.87
Average	**0.566**	**1.563**	**3.953**

3.1 Running on different channels

Table 2. Experiment *baseline* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.458	0.00	3.456
bolt	0.644	0.00	3.087
car	0.454	1.00	3.319
cup	0.763	0.00	2.488
david	0.533	2.00	1.581
diving	0.332	4.00	2.371
face	0.879	0.00	1.523
gymnastics	0.579	4.00	1.485
hand	0.625	0.00	3.560
iceskater	0.369	1.00	1.840
juice	0.794	1.00	2.640
jump	0.607	0.00	2.454
singer	0.365	0.00	0.862
sunshade	0.622	3.00	1.960
torus	0.804	0.00	1.663
woman	0.7011	1.00	1.539
Average	**0.596**	**1.063**	**2.239**

Of course, we can immediately notice that the speed is sensibly reduced. But this method slightly improves the accuracy in most of the sequences, and is much more robust: we dont lose the target anymore on the sequences bicycle, torus, as well as the most challenging sequences bolt and hand, which is a great improvement.

3.2 Scale adaptation

Table 3. Experiment *baseline* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.621	0.00	1.523
bolt	0.575	0.00	1.651
car	0.554	0.00	2.024
cup	0.744	0.00	1.653
david	0.356	1.00	1.766
diving	0.314	5.00	1.741
face	0.852	0.00	1.045
gymnastics	0.590	4.00	1.266
hand	0.272	3.00	1.697
iceskater	0.330	7.00	2.463
juice	0.879	0.00	1.565
jump	0.614	0.00	1.615
singer	0.692	0.00	1.070
sunshade	0.474	2.00	1.604
torus	0.445	1.00	1.921
woman	0.374	1.00	2.431
Average	**0.543**	**1.500**	**1.690**

This method has good and bad effects on our results. We obtain much better results (accuracy and robustness) on the sequences containing target size change: juice, singer, car, david, sunshade. On the other hand, we introduce a lot of negative effect on some other sequences: iceskater, hand, torus So in some cases, it is a great improvement, but it is too uncertain to keep it (without mentionning the execution time, once again reduced). I decided not to keep this feature.

3.3 Initial estimate of target

Table 4. Experiment *baseline* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.46	0.00	3.62
bolt	0.65	0.00	2.78
car	0.46	1.00	3.90
cup	0.76	0.00	2.53
david	0.54	1.00	2.01
diving	0.33	4.00	2.57
face	0.88	0.00	2.26
gymnastics	0.59	3.00	1.65
hand	0.62	0.00	2.70
iceskater	0.40	4.00	1.67
juice	0.64	0.00	1.71
jump	0.61	0.00	2.62
singer	0.37	0.00	1.24
sunshade	0.64	3.00	1.99
torus	0.79	0.00	2.16
woman	0.71	1.00	2.47
Average	**0.591**	**1.063**	**2.369**

The execution time objective is fulfilled, since the algorithm is now faster on most of the sequences. Concerning the other objective, the results are also good: we dont lose the target anymore in the sequence juice, and we also improved the robustness on the sequences david and gymnastics. The only bad point is on the challenging sequence iceskater, where the robustness was degraded. In conclusion, this new feature is also a good improvement.

3.4 Final results

Table 5. Experiment *baseline* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.46	0.00	3.62
bolt	0.65	0.00	2.78
car	0.46	1.00	3.90
cup	0.76	0.00	2.53
david	0.54	1.00	2.01
diving	0.33	4.00	2.57
face	0.88	0.00	2.26
gymnastics	0.59	3.00	1.65
hand	0.62	0.00	2.70
iceskater	0.40	4.00	1.67
juice	0.64	0.00	1.71
jump	0.61	0.00	2.62
singer	0.37	0.00	1.24
sunshade	0.64	3.00	1.99
torus	0.79	0.00	2.16
woman	0.71	1.00	2.47
Average	**0.591**	**1.063**	**2.369**

Table 6. Experiment *region_noise* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.46	0.00	2.70
bolt	0.68	0.60	3.22
car	0.44	0.80	4.33
cup	0.71	0.00	2.69
david	0.55	1.40	2.20
diving	0.32	3.87	2.58
face	0.76	0.00	2.25
gymnastics	0.51	3.27	1.92
hand	0.48	2.00	2.98
iceskater	0.41	3.93	1.91
juice	0.59	0.00	2.33
jump	0.59	0.00	2.74
singer	0.35	0.60	0.94
sunshade	0.62	2.27	1.87
torus	0.71	0.40	1.90
woman	0.65	0.87	1.75
Average	**0.553**	**1.250**	**2.393**

Table 7. Experiment *grayscale* results for tracker *MCDF*

	accuracy	**robustness**	**speed (fps)**
bicycle	0.44	0.00	3.54
bolt	0.67	1.00	3.98
car	0.46	1.00	4.43
cup	0.80	0.00	2.95
david	0.62	1.00	2.16
diving	0.33	4.00	2.01
face	0.88	0.00	1.76
gymnastics	0.61	2.00	1.58
hand	0.52	0.00	2.84
iceskater	0.40	4.00	1.81
juice	0.64	0.00	2.39
jump	0.60	0.00	2.95
singer	0.37	0.00	1.50
sunshade	0.57	1.00	2.44
torus	0.81	0.00	1.98
woman	0.63	1.00	1.66
Average	**0.583**	**0.938**	**2.499**

4 Discussion

During this seminar, I explored different ways to improve the DF tracker, and managed to keep the best features to globally increase the accuracy and robustness of the tracker on most of the sequences. The results are really positive and encouraging.

The main weak point of this new tracker is the execution time, since the MCDF tracker is ever slower than the original one. A parallelization of the convolution of the layers of the DF can hide this problem, but we can also imagine algorithm improvements to fasten the tracker and reach a real-time execution.

A lot of possibilities exist to continue the improvement of this tracker. We can try to apply all these new features on the Enhanced Distribution Field Tracker, already faster and more accurate than the DF tracker. Another big improvement can be done by finding a better way to adapt the size of the window. Finally, we can imagine a great improvement of speed by exploding each layer in sub-layers, small enough to be convoluted quickly, but large enough to enable an accurate gradient descent. The local optimum will be reached by small steps, each step requiring only a small and fast-to-compute sublayer.

In conclusion, I believe that this recent tracker, in addition to my proposed improvements and other possible ways to extend it, can become a robust, accurate and real-time short-tracker competing and eventually exceeding the best state-of-the-art short-term trackers.

5 References

Laura-Sevilla-Lara and Eril Learned-Miller: Distribution Fields for Tracking. In CVPR, 2012.

Michael Felsberg: Enhanced Distribution Field Tracking Using Channel Representations, VOT 2013.