# GraphTrack: Fast and Globally Optimal Tracking in Videos

Brian Amberg and Thomas Vetter
University of Basel
Department of Mathematics and Computer Science
{brian.amberg, thomas.vetter}@unibas.ch

## Abstract

*In video post-production it is often necessary to track interest points in the video. This is called off-line tracking, because the complete video is available to the algorithm and can be contrasted with on-line tracking, where an incoming stream is tracked in real time. Off-line tracking should be accurate and – if used interactively – needs to be fast, preferably faster than real-time. We describe a 50 to 100 frames per second off-line tracking algorithm, which globally maximizes the probability of the track given the complete video. The algorithm is more reliable than previous methods because it explains the complete frames, not only the patches of the final track, making as much use of the data as possible. It achieves efficiency by using a greedy search strategy with deferred cost evaluation, focusing the computational effort on the most promising track candidates while finding the globally optimal track.*

## 1. Introduction

Interactive, offline feature tracking [2, 4, 12, 13] is of great interest in movie postproduction [1], because it allows the realistic introduction of effects and virtual objects into the video. In this setting, it is necessary to track a feature through an arbitrary amount of frames, even in the presence of occlusions and when the appearance of the tracked feature varies. This cannot be solved by a fully automatic system, instead an interactive interface is needed which gives the user immediate feedback on the tracking results. This implies, that the system has to track faster than realtime, because we immediately need the track in the complete video. The higher speed requirement makes offline tracking more difficult than online tracking, but on the other hand the algorithm can reason over the whole sequence at once, while an online tracking algorithm does not have access to future frames and therefore suffers from drift [9].

Here, we build on the work of [4], who proposed an extremely fast offline tracker, which we will call DP-TRACK. DP-TRACK achieves its – faster than realtime – speed by preprocessing the video in a computationally intensive step without user interaction, and using the search structure from this preprocessing step for fast lookup of candidate matches. It then find a path through these candidates which simultaneously (1) passes through the landmarks, (2) minimizes the difference in appearance between the marked and detected patches and (3) minimizes the length of the track.

When the user sees that the track is lost in a frame, she can mark the missed patch, and the algorithm recalculates the optimal track over the whole video. This is the behavior which we are searching for, but it turns out that when the appearance of the patches changes too strongly throughout the video, the tracking becomes unstable. Correcting a bad frame can lead to loosing track in regions of the video which were tracked correctly before adding the additional input.

The reason for this behaviour, is that the appearance model becomes too broad, *i.e*. the uncertainty in the patch appearance increases. We correct this behaviour and increase the tracking stability by explaining not only the track, but instead the full video. This makes it necessary to introduce a background appearance model in addition to the appearance model of the interest points used in DP-TRACK. The effect is, that marking an interest points in a frame does not only tell us *'this is what the interest point looks like'*, but also for every patch which was not clicked upon *'this is what the interest point does not look like'*.

We formalize the optimization as a shortest path search in a suitably constructed graph. The graph search improves the efficiency of the algorithm, which allows us to incorporate the background model at a moderate additional cost.

## 2. Contributions

DP-TRACK as formulated in [4] is flawed, it does not find the global minimum of its energy function, because the occlusions are not correclty formulated in the dynamic programming optimization method. We overcome this by using a different search strategy which allows us to (1) extend the formulation with a background model, (2) focus the computation on the most promising regions and (3) find the globally optimal path given the user constraints.

## 3. Method

The algorithm finds the most probable track through the video given interest points marked in some frames. Denote the video with $\mathcal{V} = (\boldsymbol{f}_1, \ldots, \boldsymbol{f}_F)$, where $\mathcal{V}$ is a tuple of frames $\boldsymbol{f}_i$, and a track as $\mathcal{T} = (x_1, \ldots, x_F)$, where each $x_i$ is the position of the interest points in frame $i$. In $L \ll F$ frames we have marked interest points, which we denote by $\mathcal{L}$. We use Bayes' theorem to model the probability of a track given the video and the landmarks as

$$p(\mathcal{T} \mid \mathcal{V}, \mathcal{L}) \propto p(\mathcal{V}, \mathcal{L} \mid \mathcal{T}) p(\mathcal{T}) \quad . \tag{1}$$

The video determines the position of the landmarks, but given a track these two are independent. We write this as

$$p(\mathcal{T} \mid \mathcal{V}, \mathcal{L}) \propto p(\mathcal{V} \mid \mathcal{T}) p(\mathcal{L} \mid \mathcal{T}) p(\mathcal{T}) \quad . \tag{2}$$

We assume that the user has clicked correctly, such that $p(\mathcal{L} \mid \mathcal{T})$ is a Dirac distribution. This might seem like a strong requirement, but actual user input turned out to be sufficiently accurate. The extensions necessary to make the algorithm more robust to incorrectly or noisily marked landmarks are straightforward, but result in a longer runtime. Assuming correctly clicked landmarks allows us to unclutter the equations by omitting the $\mathcal{L}$ and reasoning only over tracks which pass exactly through the selected interestpoints. We will now describe how we model the prior and the video likelihood.

### 3.1. Track Prior

The prior over tracks is modelled as a Markov chain looking back a single frame, such that the position of a landmark in frame $i$ depends only on the position of the landmark in frame $i-1$

$$p(\mathcal{T}) = p(x_1) \prod_{i=2}^{F} p(x_i \mid x_{i-1}) \quad . \tag{3}$$

Using only the previous frame, we are restricted to a first order motion model, which assumes that the new position is probably close to the old position. We therefore model the new position as an isotropic Gaussian centered on the position in the previous frame. For the first frame we are using a uniform distribution, (which is in this case a proper distribution, because we are considering only a discrete set of positions).

$$\begin{aligned} p(x_0) &\propto 1 \\ p(x_i \mid x_{i-1}) &\propto \exp\{-\lambda_d \|x_i - x_{i-1}\|^2\} \quad . \end{aligned} \tag{4}$$

### 3.2. Video Likelihood given the Track

We assume that the likelihood factorizes into a term for each frame and pairs of adjacent frames

$$p(\mathcal{V} \mid \mathcal{T}) = \prod_i p_1(\boldsymbol{f}_i, \mid x_i) \prod_{i=1}^{F-1} p_2(\boldsymbol{f}_i, \boldsymbol{f}_{i+1} \mid x_i, x_{i+1}). \tag{5}$$

The per-frame factor $p_1$ is learned from positive example patches $\mathcal{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_L)$ and negative example patches $\mathcal{N} = (\boldsymbol{n}_1, \ldots, \boldsymbol{n}_N)$ from the frames where interest points are marked. We denote the class of interestpoints as $\mathrm{ip}$ and the background class as $\mathrm{bg}$, such that we can write the resulting appearance model as

$$\begin{aligned} p(\boldsymbol{f}_i(x) \mid \mathrm{ip}) &\propto \exp\{-\lambda_f \min_j \|\boldsymbol{f}_i(x) - \boldsymbol{p}_j\|^2\} \\ p(\boldsymbol{f}_i(x) \mid \mathrm{bg}) &\propto \exp\{-\lambda_b \min\{\delta_b, \min_j \|\boldsymbol{f}_i(x) - \boldsymbol{n}_j\}\|^2\} \end{aligned} \tag{6}$$

where $\boldsymbol{f}_i(x)$ is the patch at position $x$ in frame $\boldsymbol{f}_i$, and $\delta_b$ is a parameter giving the maximum possible distance from the background. $\delta_b$ is used to model the fact that new samples which are very far from both the known interest points and the known background samples are more probably background than interest points. The norm used here is the Euclidean distance between feature vectors extracted at each patch. The details are given in section 7, but are not necessary to follow the algorithm. Assuming independence beween the patches in a frame[1] allows us to factor the per-frame term given the track as

$$\begin{aligned} p_1(\boldsymbol{f}_i \mid x_i) &= p(\boldsymbol{f}_i(x_i) \mid \mathrm{ip}) \prod_{x \neq x_i} p(\boldsymbol{f}_i(x) \mid \mathrm{bg}) \\ &= \frac{p(\boldsymbol{f}_i(x_i) \mid \mathrm{ip})}{p(\boldsymbol{f}_i(x_i) \mid \mathrm{bg})} \prod_x p(\boldsymbol{f}_i(x) \mid \mathrm{bg}) \quad . \end{aligned} \tag{7}$$

The product in Eqn. 7 is independ on the choice of $x_i$, allowing us to simplify Eqn. 7 to

$$p_1(\boldsymbol{f}_i \mid x_i) \propto \frac{p(\boldsymbol{f}_i(x_i) \mid \mathrm{ip})}{p(\boldsymbol{f}_i(x_i) \mid \mathrm{bg})} \quad . \tag{8}$$

In the pairwise appearance term we encode the assumption that the appearance of the tracked patch changes only gradually between frames, such that

$$\begin{aligned} &p_2(\boldsymbol{f}_i, \boldsymbol{f}_{i+1} \mid x_i, x_{i+1}) \\ &\quad \propto \exp\{-\lambda_s \|\boldsymbol{f}_i(x_i) - \boldsymbol{f}_{i+1}(x_{i+1})\|^2\} \quad . \end{aligned} \tag{9}$$

As opposed to the per-frame factor $p_1$ which explains the complete frame, our definition of $p_2$ only explains the appearance change of the patches under the tracked patch, and

[1] A common assumption necessary to arrive at an efficient algorithm but nonetheless a simplification, as an image with independent pixels would be very boring

ignores the remaining area of the frames. This is necessary to keep the algorithm efficient, but might be an interesting starting point for further research.

## 4. Efficient Optimization

We minimize the negative log of the posterior track probabilities, which has the form

$$
\begin{aligned}
-\log p(\mathcal{T} \mid \mathcal{V}, \mathcal{L}) &= \text{constant} \\
&+ \sum_{i=1}^{F} (\lambda_f \min_j \|\boldsymbol{f}_i(x_i) - \boldsymbol{p}_j\|^2\}) \\
&- \sum_{i=1}^{F} (\lambda_b \min\{\delta_b, \min_j \|\boldsymbol{f}_i(x_i) - \boldsymbol{n}_j\|^2\}) \\
&+ \lambda_s \sum_{i=1}^{F-1} \|\boldsymbol{f}_i(x_i) - \boldsymbol{f}_{i+1}(x_{i+1})\|^2 \\
&+ \lambda_d \sum_{i=1}^{F-1} \|x_i - x_{i-1}\|^2 \quad .
\end{aligned} \tag{10}
$$

DP-TRACK uses a dynamic programming approach to minimize this cost. This dynamic programming method fails to find the global optimum when handling occlusions, as is demonstrated in the appendix. For our method we observe that the cost function can be encoded as a graph and minimized efficiently and globally optimal with a shortest path search. We use a version of Dijkstra's shortest path search [6] which speeds up the search and allows us to efficiently incorporate our background model.

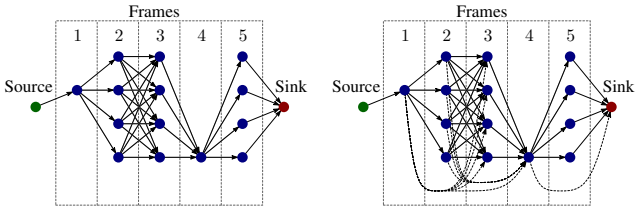
We now describe how to map the cost to a graph. We can interpret the cost as a directed acyclic graph. This graph has one layer for every frame, and in each layer a node $n_{i,x}$ for every patch. Each node has a weight corresponding to the single frame match

$$
w(n_{i,x}) = -\log p_1(\boldsymbol{f} \mid x) \quad . \tag{11}
$$

The nodes of one frame are connected to the nodes of the next frame by edges $(n_{i,x_i}, n_{i+1,x_{i+1}})$ with a weight consisting of the movement prior and the pairwise appearance term

$$
\begin{aligned}
&w(n_{i,x_i}, n_{i+1,x_{i+1}}) \\
&\quad = \lambda_s \|\boldsymbol{f}_i(x_i) - \boldsymbol{f}_{i+1}(x_{i+1})\|^2 + \lambda_d \|x_i - x_{i-1}\|^2
\end{aligned} \tag{12}
$$

The frames where the interest points were marked contain only a single node at the interest point. This ensures that all paths run exactly through the selected points. The graph has two additional special nodes, the source, which is connected to all patches of the first frame, and the sink, which is the target of all patches of the last frame. See Fig. 1 for a small example. Every path from source to sink passes through exactly one node of every layer, as each node is only connected to the directly following layer. The possible paths map therefore one-to-one onto the possible tracks, and the sum of the costs of all edges and nodes of a paths is exactly equal to the cost assigned to the corresponding track. The shortest path from source to sink is therefore also the global minimum of the cost function.




Without occlusion handling With occlusion handling

Figure 1. The cost function can be interpreted as a directed acyclic graph with weights on the nodes and edges. The optimal track is the path from source to sink which has the minimal weight. On the left is a graph for the cost function without occlusion handling, and on the right the same graph with occlusion handling. Each frame has 4 candidate patches, and frame 1 and 4 do have a marked interest point.

We search for the shortest paths using a variant of Dijkstra's shortest path using a Fibonacci heap [7]. Dijkstra's algorithm partitions the nodes into an 'active' and a 'solved' set. The minimal distance towards the nodes in the solved set is known, and for the nodes in the active set an upper bound on the distance is kept. An invariant of the algorithm is that each node in the active set has a distance which is larger than the maximal distance to any node in the solved set. This invariant is maintained, by greedily choosing the node from the active set which has the minimal distance, moving it to the solved set and updating the distance bound towards the descendants of the expanded node with the distance of the path through the expanded node. To do this efficiently we need a priority queue with a $O(1)$ decrease key operation. The full algorithm is:

1. The source is put into a priority queue, with an associated track length of zero.
2. The remaining nodes are inserted into the priority queue with an associated track length of $\infty$.
3. Iterate, until the sink is expanded:
   A. Select the node $x_i$ from the queue, which has the minimal track length.
   B. Expand $x_i$ by doing for every descendent $x_j$ of $x_i$:
      a. Calculate the length of the shortest path to $x_j$ passing through $x_i$, this is the distance towards $x_i$ plus the edge cost $w(x_i, x_j)$ plus the per-frame cost $w(x_j)$.

b. If this length is smaller than the length associated with $x_j$, decrease the length of $x_j$ to the new length, set a parent pointer from $track_j$ to $track_i$ and update the priority queue.

4. Follow the parent pointers from the sink to the source to recover the shortest path

While this optimization is efficient, it is still slow to calculate the weights for all edges and nodes. Especially evaluating Eqn. 7 for each node is expensive, because this involves a search over a large number of negative examples, and the graph has a huge number of edges, as every frame is densely connected to the following frame. We can overcome this by observing that, for realistic settings of the parameter weights, we only have to expand a small fraction of the nodes. The cost of the shortest path towards most nodes is larger than the cost of the minimal path towards the sink. This is especially true when the track prior favours small movements, *i.e.* $\lambda_d$ is relatively large. In step 3.ii we have to calculate for all descendents of the expanded nodes the edge weights. This requires $N_{\text{Expanded Nodes}} N_{\text{Candidates/Frame}}$ evaluations. We will now describe a method to lazily evaluate the edge costs such that only a few of the *shortest edges* going into the expanded nodes are evaluated, bringing these expensive operations down to the order of $O(N_{\text{Expanded Nodes}})$.

Note that while we are, for a strong motion prior, effectively calulating costs only for a few nodes around the currently best track, this is different from methods which restrict the search to a small region around the current tracked position, as we are still searching over the full frame whenever the optimal track requires this. The search is only arranged in the most efficient order.

To be able to lazily evaluate the appearance cost we modified Dijkstra's algorithm by using a lower bound on the track length instead of the actual track length. This lower bound is calculated by replacing the per-frame cost $w(x_j)$ in step 3.B.a with a lower bound of the frame cost if $x_j$ has not been expanded before. The actual cost of $x_j$ is then calculated when it is expanded. Updating the lower bound of the frame cost can lead to it being no longer the overall smallest node. In that case, instead of further expanding it, we just add it back into the priority queue. This guarantees that the globally optimal path is found while performing only the absolutely necessary amount of computation. As every lower bound is updated only once, we can guarantee that the number of times a node is taken from the priority queue is maximally doubled.

The approximate frame cost is calculated by using a restricted set of background examples. We include for each positive example the one negative example which is closest to it.

The true per-frame cost is the distance towards the closest positive example minus the clamped distance towards the closest negative example plus the unknown constant from from Eqn. 7. But the constant was left out, as it does not change the position of the minimum. This can result in a negative distance, which is not solveable with Dijkstra's algorithm. To overcome this we add to all nodes an upper bound on the distance to the background. The upper bound is found by taking the distance between each candidate patch and the recduced background model. This does not change the minimal path, because every path has to go through the same number of nodes, and the offset is added to all nodes.

While the algorithm as presented minimizes the number of nodes which actually need to be expanded, it is still too slow to be used with all pixels in all frames. Instead, we select in a first step a number of candidate patches from each frame (typically 150-250 patches) which are the input for all stages of the algorithm. The efficient selection of candidates is explained in section 6.

## 5. Occlusions

So far, our algorithm cannot handle occlusions. The occlusion handling method introduced in [4] does not result in a globally optimal solution (as demonstrated in the appendix, but we present a relatively efficient globally optimal occlusion handling method here.

To model occlusions we introduce a new binary random tuple, $\mathcal{O} = (o_1, \ldots, o_F)$ describing the occlusion state of each frame. We find the MAP estimate of

$$p(\mathcal{T}, \mathcal{O} \mid \mathcal{V}, \mathcal{L}) \propto p(\mathcal{V}, \mathcal{L} \mid \mathcal{T}, \mathcal{O}) p(\mathcal{T}) p(\mathcal{O}) \quad , \qquad (13)$$

where we have assumed that occlusion and track movement are independent. We model the occlusion prior again as a Markov chain

$$p(\mathcal{O}) = p(o_0) \prod_{i=1}^{F} p(o_i \mid o_{i-1}) \qquad (14)$$

The resulting cost could be directly encoded in the graph by adding for every pixel in every frame a node corresponding to the state of being occluded at this position. That is not feasible, as the number of nodes would be too large. An efficient method can be found by noting that the track position during a run of occluded frames depends only on the motion model, and with our first order motion model is completely determined by the endpoints of the occluded run. We can therefore combine all possible occluded subpaths between two frames into a single edge, whose weight is the minimum of all these paths. This removes all occluded nodes, and instead introduces on the order of $\mathcal{N}^2_{\text{frames}} N_{\text{Candidates Per Frame}}$ edges. The resulting path search is still relatively efficient, because the weights on all these edges are only calculated

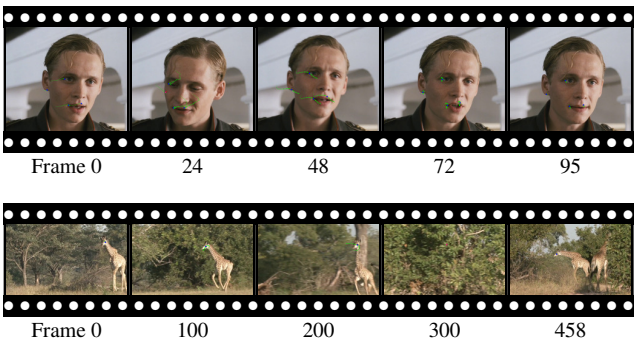



Figure 2. Tracking results on a talking head sequence and on a video of a giraffe. Between one and three user clicks were needed to achieve accurate tracking for the head sequence. Note the correct handling of the occluded ear, which was achieved with a single click. The eye of the running giraffe needed eight clicks, of which three marked occlusions. Please refer to the accompanying video for more details.

lazily for the expanded nodes. For long sequences with more than 100 frames, the cost of updating all nodes in the subsequent frames becomes substantial. In this case we propose to limit the number of edges by adding only edges from a node to the 10 directly following frames, then to every second frame in the range 11-20, every third in the range 21-30 etc. This assumes that the expected length of occlusions is relatively short, or that the object of interest is visible in large parts of the video. For long occlusion we might find a track which stays occluded longer than necessary. This can be remedied by the user, by marking the track position in the first non-occluded frame.

Also, we speed up the update of the children by checking if the current cost plus the occlusion cost is already larger than the current minimal cost of a child. In that case we can skip the relatively more expensive calculation of the similarity cost.

Instead of specifying the four probability values of $p(o_i \mid o_{i-1})$ we are using two costs, an occlusion start cost and a cost between occluded frames, such that the cost associated with an occlusion edge from $x_i$ to $x_j$ is

$$\underbrace{\lambda_o + \lambda_r(j-i-1)}_{\text{occlusion}} + \underbrace{\frac{\lambda_s \|\boldsymbol{f}_i(x_i) - \boldsymbol{f}_j(x_j)\|^2}{j-i-1}}_{\text{similarity}} + \underbrace{\frac{\lambda_d \|x_i - x_j\|^2}{j-i-1}}_{\text{distance}}, \tag{15}$$

where we assumed that the appearance and position change linearly during the occlusion.

## 6. Candidate Selection

To make the algorithm faster than real time it is necessary to restrict the number of candidate positions taken into consideration in each frame. This section explains, how we efficiently extract candidate positions from the video. The GRAPH-TRACK search algorithm is able to handle a few hundred candidates per frame efficiently, as opposed to the four candidates per frame used by DP-TRACK.

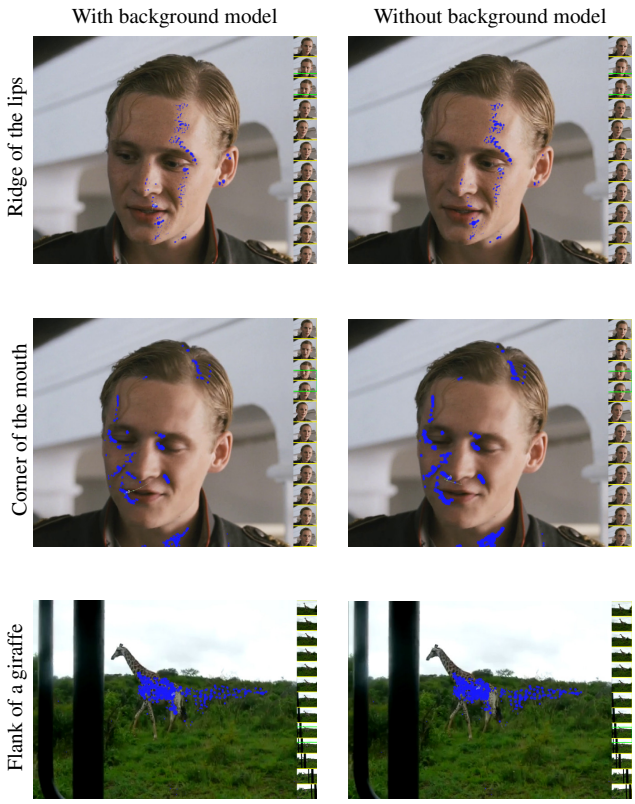



Figure 3. The candidate points which were selected in a single frame of a video sequence. No spatial prior is imposed when selecting the candidate points, such that our algorithm can handle arbitrary large movements if the evidence is strong enough. We show the matches for the ridge of the lip, the right eye corner and the flank of a giraffe. The size of the dots corresponds to $p(\boldsymbol{f}_i \mid track_i)$, where the left column is using the background model and the right column is not using the background model. We picked three examples where the correct track (shown as a green line) is only found when including the background model. In all sequences the landmark was marked in the first frame.

Finding the candidates is essentially a template matching problem, and could be addressed by methods such as [3, 11, 8, 5], but for offline tracking it is allowable to invest some time into the preprocessing, if this leads to immediate feedback during the user interaction. Following [4] we therefore preprocess the image such that each patch is represented by a feature vector. The details of the feature extraction are given in section 7, for now it suffices to say that we extract 16 one-byte features at each pixel position in a one-off preprocessing phase. The rest of the algorithm then works with this 16 byte representations of the image patches, allowing much faster calculations than those ob-

tainable with a generic template matching method.

To efficiently select candidates, [4] stored the patch features into a KD-Tree structure using 24 bytes per patch. While the KD-Trees are quite efficient we found that an exhaustive search can be as fast, but requires only $2/3$ the amount of memory allowing the handling of 1.5 times longer sequences. Another advantage is that with the efficient feature extraction described in section 7 and without the need to construct the KD-Trees the preprocessing time dropped from hours ([4], personal communication) to minutes. The candidates are chosen to be the $N_{\text{candidates}}$ patches which are closest to the positive examples and are locally minimal. To make the search efficient we are not calculating the sum of squared differences while deciding on the candidates, but instead use the sum of absolute differences as a proxy function. The sum of squared differences cost is then calculated only for the examples selected based on the proxy function. The SSE instruction set of modern CPUs contains a command to calculate the sum of absolute differences between two pairs of unsigned eight byte vectors in a single instruction. By organizing the data accordingly we can efficiently calculate the differences. While computing the differences we are performing a non minimum suppression. This requires access to the costs of the current image row and the last row. This fits into the processor cache, resulting in a fast algorithm. Already during the run we are choosing the top $N_{\text{candidates}}$ positions with a heap data structure.

Additionally, all candidates with a feature space distance of more than 72 from the positive examples are rejected. The threshold of 72 was selected manually once on some test sequences, and proved to work well for all other videos.

## 7. Feature Extraction

The algorithm consists of an offline phase and an online phase. In the offline phase we calculate a 16 byte feature vector for every patch in the input video. The features are independent from the track, such that the offline calculations have to be done only once per video. We use the features proposed in [4], where a linear filter jet [10], is used, which is adapted to the video under consideration. The features are a projection of the patches $p_i$ into a PCA basis of all patches of the video:

$$\boldsymbol{f}(x_i) = \boldsymbol{U}^T(p_i - \bar{p}) \quad . \tag{16}$$

Here $p_i - \bar{p}$ are the patches centered by the mean of all patches in the video. We use the 16 basis vectors of the PCA which have the largest associated eigenvalues and explain most of the variability observed in the video. The features are then scaled to the range of $[0, 255]$ and quantized to eight bits. The PCA basis is trained on the video, by sampling 1/8th of the patches of every frame.

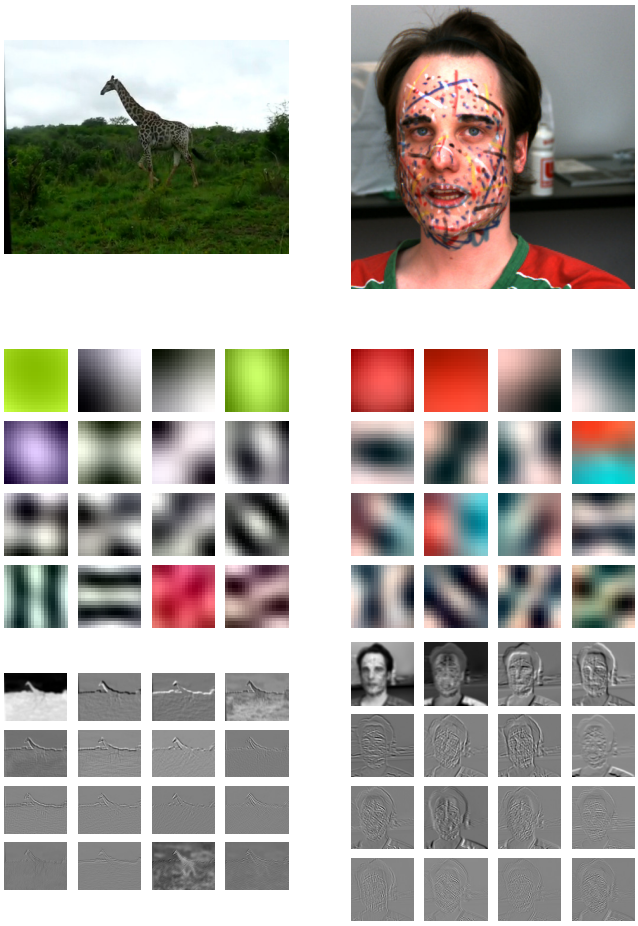


Figure 4. The filter bank responses for two two different videos. A frame of each of the videos is shown next to a depiction of the video specific filter jet, offset such that it can be visualized in RGB space. At the bottom of the figure we also show the response of the features on the example frame. While the spatial structure of the jets is similar for both videos, they do differ a lot in the color distribution. Filtering the video with a specially tuned basis decorrelates the patches, and thereby removes some of the redundancy in the colour channels, without losing the available information as a transformation to grayscale would.

The feature extraction runs at about 20 frames/minute on a commodity PC for a VGA-sized video, when exploiting the following observations. The average of all patches of a video has a constant color, because (assuming circular boundary conditions) every pixel in the video occurs exactly once at every position in one of the patches. Therefore instead of calculating the average patch, it is sufficient to calculate the average color, and subtract it from the input video. The patches of the resulting video are then already mean-centered. This turns the projection into the PCA basis into a convolution of the mean-centered frames with the basis vectors. This is implemented efficiently using FFT, by using the well known fact that a convolution can be ex-

pressed as $A * B = \mathcal{F}^{-1}(\mathcal{F}(A) \circ \mathcal{F}(B))$, where $\circ$ denotes the elementwise product. As we need to convolve all images $A_i$ with all kernels $B_j$, we can calculate the forward transform of all kernels and images, and then get the outputs by elementwise multiplication and a backward transform of all combinations $(A_i, B_j)$.

The filter jet extracted from two different videos is shown in Fig. 4, showing that the basis adapts to the video, such that it captures as much information as possible.

## 8. User Interface

While [4] proposed to let the user modify the values of the parameters, we found that when working with similar scenes it is faster to supply a few additional interest points and use a preset of parameters for the scene type. Now that we have included a background model we found that the result of adding a landmark is more predictable than that of changing the priors, and it is easier to teach new users the meaning of selecting landmarks than the exact meaning of the priors. We found that for talking heads the default parameters $\lambda_f = \lambda_b = 1$, $\lambda_s = 10$, and $\lambda_d = 10$, $\lambda_o = 5000$, $\lambda_r = 1.5\lambda_o$, $\delta_b = 4096$ gave good results over most sequences. Nonetheless different priors are needed for scenes of differing character, e.g. the giraffe sequence shown in Fig. 2 has faster motion and more occlusion, and therefore needed a smaller occlusion and motion weight.

## 9. Refinement

As in DP-TRACK we follow the path search with a refinement step, where the best SSD match of the $15 \times 15$ image patches within 8 pixels of the track to the image patches of the positive examples is found.

## 10. Experiments

**Accuracy:** In Fig. 2 we show example tracks for a talking head sequence with rapid movements and pose changes, and a video of running giraffe with severe occlusions. See also the accompanying videos to get a feeling for the user interaction required to mark up these videos. As the process is interactive, and we are using a background model, we are able to track anything in any video, given enough user input.

**Speed:** To get a feeling for the speed of the algorithm we evaluated the calculation time for varying numbers of landmarked frames (which increase the size of the foreground model, but also simplify the search graph), and for varying numbers of candidates. We did this by marking a 107 frame sequence with more landmarks than absolutely necessary and then chose all possible subsets of the marked landmarks. The experiments were done for search graphs with different maximum occlusion lengths. The results are

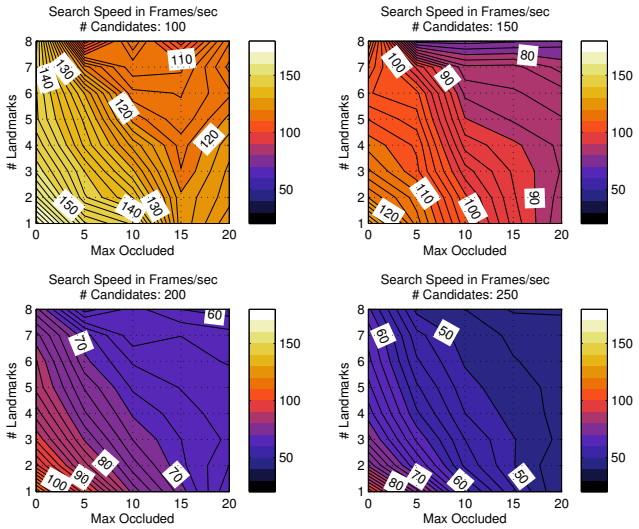


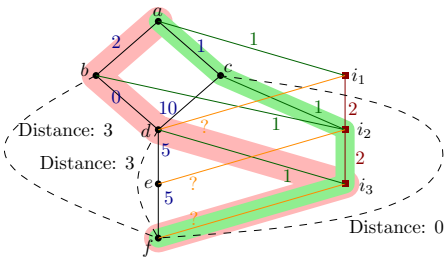

Figure 5. Tracking speed as a function of the algorithms parameters. The tracking time of GRAPH-TRACK is increasing approximately linearly with the number of landmarks, the number of candidate patches and the maximum occlusion length, where the largest influence is due to the number of candidate patches.

shown in Fig. 5. For some combination of landmarks we do not find the correct tracks. In this specific example we do need four landmarks to accurately track the eye corner during all blinking events and head pose changes. The failed tracks are included in the timing and are still interesting, as it is important that the user gets immediate feedback while marking up the video, such that she knows which frames require further attention.

As opposed to the 4 candidate patches per frame and marked interest point which were used in DP-TRACK, we are extracting between 100 and 300 candidates per frame, which makes the algorithm much more reliable on difficult frames. Our algorithm seems to scale approximately linearly in the number of frames, linearly with a small coefficient in the number of landmarks and linearly in the maximum number of occluded frames which we consider. The sweet spot seems to be at 150 to 200 candidates per frame, and with a value for the maximal occlusion length which closely matches the actual occlusion length in the video.

## 11. Conclusion

We presented an enhancement of the tracking algorithm of [4], which is more reliable on difficult videos, and more stable when the user adds additional patches. Better stability is achieved by modelling the full frame appearance, instead of only the patch appearance. Even though that seems to imply a lot more computational work, we proposed an efficient search algorithm which lazily performs the expensive computations. The beauty of our search approach is that it adapts automatically to a stronger smoothness or oc-




The image space distances between the nodes of the graph are:

| | b | c | d | e | f |
|---|---|---|---|---|---|
| a | 2 | 1 | 10 | 10 | 10 |
| b | | | 0 | 10 | 3 |
| c | | | 10 | 10 | 0 |
| d | | | | 5 | 3 |
| e | | | | | 10 |

Figure 6. The computationally less intensive occlusion handling of DP-TRACK does not find the global minimum. This figure shows the graph of a counterexample. Shown is a graph with visible nodes $a, \ldots, f$ in five frames plus the invisible states $i_1, \ldots, i_3$ of the not landmarked frames, sink and source were ommited. The dashed lines are not edges, they annotate the distance cost which would be incurred between the connected nodes. The weight of the orange lines leading out of the invisible states corresponds to this distance cost for the node which is currently connected via parent pointers along the invisible nodes. The optimal track is marked with green, while the track found by DP-TRACK is marked in red.

| Frame: 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| $a = 0/\text{source}$ | $b = 2/a$<br>$c = 1/a$<br>$i_1 = 1/a$ | $d = 2/b$<br>$i_2 = 2/c$ | $e = 7/d$<br>$i_3 = 3/d$ | $f = 6/i_3$ |

Figure 7. The DP-TRACK dynamic program corresponding to the graph in Fig. 6. Each entry contains the name of the node, the total cost up to this node and the pointer to the parent node.

clusion prior, only performing as much computation as necessary, without absolutely restricting the search. If the patch appearance gives enough evidence, arbitrarily large jumps in space and time are possible, while still being efficient.

The algorithm is useful as building block in many applications. To encourage its use we publish the source code and binaries for this project.[2]

## Appendix: Why the occlusion reasoning of DP-TRACK misses the global optimum

Here we give an example where DP-TRACK fails to find the globally optimal solution for a video when occlusion handling is used. But we start with a word of warning: [4] are very terse on the topic of occlusion handling, so our understanding might misrepresent the authors ideas. DP-TRACK does not fully exploit the graph structure of the problem, only the fact that the corresponding graph is layered, but there is a graph equivalent to the construction in DP-TRACK. This equivalent graph handles occlusions with a single additional node per frame, corresponding to the 'occluded' state. The cost of entering this node is fixed, the cost of going from an occluded state to the occluded state of the next frame is another constant, and the cost of leaving the occluded state and entering a non occluded state $x_j$ in the following frame depends on the distance between the node $x_i$ which is found by following the parent pointers along the current occlusion state and $x_j$. This greedy choice can lead to suboptimal results. To demonstrate the problem consider the graph shown in Fig. 6 and assume additionally that the between frame similarity and appearance cost are zero everywhere. The cost of going into an invisible state is set to 1 and the cost of staying in an invisible state is 2.

The optimal path through the graph in Fig. 6 is $a, c, i_2, i_3, f$, with a total cost of four, while the path found with DP-TRACK is $a, b, d, i_3, f$, with a total cost of 6. The result of applying DP-TRACK is given in Fig. 7.

[2]The source code can be downloaded from `www.cs.unibas.ch/personen/amberg_brian/graphtrack/`.

## References


[1] 2D3. Boujou. `http://www.2d3.com/`.

[2] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. In *SIGGRAPH '04*, volume 23, issue 3, pages 584–591. ACM, 2004.

[3] R. Anderson and H. Schweitzer. Fixed time template matching. In *SMC 2009, Systems, Man, and Cybernetics*, pages 1359 –1364, Oct. 2009.

[4] A. Buchanan and A. Fitzgibbon. Interactive Feature Tracking using K-D Trees and Dynamic Programming. In *CVPR '06*, pages 626–633. IEEE, 2006.

[5] L. Di Stefano and S. Mattoccia. Fast template matching using bounded partial correlation. *Machine Vision and Applications*, 13:213–221, 2003.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.

[7] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *JACM: Journal of the ACM*, 34(3):596–615, July 1987.

[8] T. Kawanishi, T. Kurozumi, K. Kashino, and S. Takagi. A fast template matching algorithm with adaptive skipping using inner-subtemplates' distances. *Int. Conf. on Pattern Recognition*, 3:654–657, 2004.

[9] L. Matthews, T. Ishikawa, and S. Baker. The template update problem. *PAMI*, 26(6):810 –815, June 2004.

[10] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *PAMI*, 19(5):530–535, 1997.

[11] H. Schweitzer, J. Bell, and F. Wu. Very fast template matching. In *ECCV '02*, volume 2353 of *LNCS*, pages 145–148. Springer, 2006.

[12] J. Sun, W. Zhang, X. Tang, and H.-Y. Shum. Bi-directional tracking using trajectory segment analysis. In *ICCV '05*, pages 717–724. IEEE, 2005.

[13] Y. Wei, J. Sun, X. Tang, and H. Y. Shum. Interactive offline tracking for color objects. In *ICCV '07*. IEEE, 2007.