Combining an Optical Flow Feature Detector with Graph-Based Segmentation *

Martin Stubenschrott
e0125672, E934

April 24, 2007

Institute of Computer Aided Automation
Pattern Recognition and Image Processing Group (PRIP)
of the
Vienna University of Technology

Assisted by

O.Univ.Prof. Dr. Walter G. Kropatsch

and

Dr. Yll Haxhimusa

*Partially supported by the Austrian Science Fund under grant FWF-P18716-N13. This work is part of the *twist* project (Tracking with Structure in Computer Vision - http://www.prip.tuwien.ac.at/Research/twist/twist.php)

Abstract

Object tracking is the complex task to follow a given object in a video stream. This paper describes an algorithm which combines an optical flow based feature tracker with color segmentation. The aim is to build a feature model and reconstruct lost feature points when they are lost due to occlusion or tracking errors.

These feature points are tracked from one frame to another with the Lucas & Kanade optical flow algorithm. Additionally, we segment each frame with the Felzenszwalb-Huttenlocher graph-based segmentation algorithm. Optical flow and segmentation are then combined to track an object in a video scene. By using this strategy, also occlusion and slight rotation or deformation can be handled.

The tracker is then evaluated on an artificial video sequence with moving balls but also on real-world sequences of a moving person. For all video sequences, ground truth data is available and compared to our results.

1 Introduction

Image segmentation [FH04, MMTB+06, HK03, HK04] and optical flow [HS80] are two very common tasks in image processing. Image segmentation partitions an image into visually distinct region. For this to work well, it is preferable to have regions of similar color and/or texture. Finding the optical flow of two images can be interpreted as finding the most probable pixel position of the first frame in the second frame. Unfortunately, this requires just the opposite of image segmentation: The more unique some small part of the image is, the better it can be matched in the next frame.

Thus, optical flow and image segmentation complement each other: Optical flow has problems with homogeneous regions, which can be handled well by image segmentation. However, segmentation has problem with fuzzy borders which can be handled by optical flow.

This paper provides a framework for a feature-based object tracker which is improved by segmentation information. Features are tracked from one frame to the next with an optical flow algorithm. While the actual optical flow can work quite well without segmentation, segmentation information helps following the entire object instead of just a few single feature points of it. Also segmentation information is essential, when we need to reconstruct some lost feature points.

Combination of these two basic image processing algorithms is not new. Indeed one motivation for this work was the paper *Exploiting Texture-Motion Duality in Optical Flow and Image Segmentation* by Michael G. Ross [Ros00]. It uses flow information to provide better segmentation results and vice versa. While this paper combines optical flow and segmentation, it is rather focused on improving segmentation results than in object tracking. Jeongho Shin et al. proposed an optical-flow based feature tracker algorithm which could track non-rigid objects in real world scenes. Their experiments showed that a feature based object tracker can work well, however they did not take segmentation information into account [SKK+05]. Therefore their tracked „objects" are rather tracked points, our work tries to track a fully outlined object.

1.1 Object tracking methods

Current object tracking approaches can be roughly categorized into five main classes depending on the target representation [CSE05]: Model-based, appearance-based, contour-based, feature-based and hybrid methods.

Model-based object tracking needs a priori knowledge of the objects' shape. This can work well for very specific tasks, but is not extendible for general scenes.

Appearance-based techniques track objects by the appearance of the connected region, which may include color or texture information. This approach has problems with deformations, occlusion or rotation of the object.

Contour-based methods usually track only the outline of the object, which reduces computational load, but still has similar problems as *appearance-based* methods.

Feature-based tracking uses features of a video object to track part of if. The problem of this approach is grouping these features together to determine which of them belong to the same object. Our proposed algorithm falls into this tracking class.

As usual, there is not a strict border between these techniques, and therefore [CSE05] denotes a *hybrid-based* approach as the fifth large group of current object tracking methods.

1.2 Goals

The goal of this work is to examine different ways how an optical-flow based feature tracker can be combined with segmentation. The result should be a good object tracker with high-level segmentation.

For this to work, we need to put effort on a basic outlier detection which discards wrongly detected feature points. Those need to be reconstructed in a lost feature point restoration process. Also an occlusion detection system is part of my goals, which finds an even fully occluded object, once it appears again.

1.3 Overview

We begin in Section 2 with a short description, how our segmentation algorithm works. In Section 3, we give an overview of the optical flow calculation. With this knowledge, we are ready to give a detailed description of how these two techniques are combined in Section 4, where we create an object tracker. This tracker is later evaluated in Section 5 with an artificial and with a real-world video sequence.

2 Segmentation

The purpose of segmentation is to cluster visually similar, neighboring regions together. It is quite difficult, if not impossible, to find one perfect segmentation which is neither too coarse, nor too fine for all applications.

The segmentation is done with the Felzenszwalb-Huttenlocher segmentation algorithm [FH04] which belongs to the class of graph-based segmentation algorithms. Its runtime efficiency is $O(n \log n)$ for n image pixels. A sample segmentation can be seen in Figure 2b. Each segment is colored randomly and transparently overlayed over the original image. In this example, the person at the bottom-right of the image is segmented very well, but there are some minor patches on the floor, which should not be there in a perfect segmentation.

Graph-based techniques use a graph $G = (V, E)$ with vertices $v_i \in V$ representing pixels of an image, and $(v_i, v_j) \in E$ representing the edges between neighboring pixels. Each edge has a weight $w((v_i, v_j))$, which measures the (dis-)similarity between neighboring image pixels. This weight is usually obtained by the color differences between pixels, in our case the Euclidean distance of the red, green and blue color values is used. Now the idea is to create connected components $C_1, \ldots, C_k$ which consist of edges with low weights (=look similar) and have high weights (=strong boundaries) to other connected components. Felzenszwalb and Huttenlocher therefore introduce three concepts:

- Internal difference of a component $C \subseteq V$

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

  $Int(C)$ is the largest weight within a component (MST denotes the minimum spanning tree of this component)

- Difference between two components $C_1, C_2 \subseteq V$

$$Dif(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2, (v_1, v_2) \in E} w((v_1, v_2))$$

  Dif is the minimum weight edge connecting C_1 and C_2 or ∞ if there is no such edge

- Minimum Internal Difference of two components $C_1, C_2 \subseteq V$

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2))$$

  $\tau(C_i)$ is a threshold function whose value decreases as the component gets larger. Usually it is defined as $\tau(C_i) = \frac{k}{|C_i|}$, where k is a constant factor, and $|C_i|$ the size of the component in pixels.

Using these concepts the segmentation algorithm can be defined as:

1. Sort edges by non-decreasing edge weight
2. Start with an initial segmentation, where each vertex v_i is in its own component C_{v_i}
3. v_i and v_j are the vertices, which are connected by the edge with the lowest weight. If v_i and v_j are already in the same component, then continue. Otherwise merge the two components where v_i and v_j belong to, if and only if $MInt(C_{v_i}, C_{v_j}) \geq Dif(C_{v_i}, C_{v_j})$.
4. Repeat step 3 for all other edges in order

While this algorithm works well for artificial images, for real world examples with noise and other small artifacts, results can be greatly improved by some simple pre- and postprocessing steps:

- Preprocessing is done with a Gaussian filter which smooths the image to remove artifacts.
- Postprocessing is done by merging small components to its neighboring component. The minimum component size m can be adjusted.

3 Optical Flow

Optical flow was defined by Horn and Schunck as the distribution of apparent velocities of movement of brightness patterns in an image [HS80]. While this method can provide a high density of velocity vectors by minimizing a *global* energy function, it is very vulnerable to noise. In our experiments, we could achieve better results with the *local* Lucas & Kanade [LK81] method which can handle noise much better.

Instead of calculating a flow for the whole image like the Horn & Schunck algorithm does, this method calculates a pixel displacement vector for a single pixel. More formally, for a given pixel u in an image I, we find the corresponding location $v = u + d$ in the new image J. d is the displacement vector, which can be calculated by minimizing the error function:

$$\epsilon(d) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x + d_x, y + d_y))^2$$

ω_x and ω_y are integers which define the size of the integration window, where the flow vectors are calculated.

For best results, a pyramidal implementation [Bou03] was chosen. More exactly, a regular pyramid where each level is 1/4 of the size of its previous level. Therefore the pyramid with 4 levels of an image I of size 640×480 consists of 5 images I^0, I^1, I^2, I^3 and I^4 with sizes 640×480, 320×240, 160×120, 80×60 and 40×30.

The error function is first calculated for the deepest pyramid level L_d. The result serves as an initial guess for the next level $L_d - 1$, where the calculation is performed again. This is done until we reach the finest level L_0 of the image. Using a pyramidal approach has the advantage that the actual integration window size can be kept quite small, but also larger motions can be handled well.

The actual calculation for each pyramid level is done by calculating a spatial gradient matrix G of the image derivatives I_x and I_y. Now the perfect optical flow vector for this level d^L of G is calculated in an iterative Newton-Raphson fashion. For the exact mathematical formulation, have a look at [Bou03].

It is important to note that all computations are done on subpixel basis, which yields better results over time. The image brightness for subpixels is calculated using bilinear interpolation.

4 The tracking algorithm

In Section 2 we have explained how we perform the image segmentation with the Felzenszwalb & Huttenlocher algorithm. Section 3 covered the calculation of feature points from one frame to another with the Lucas & Kanade optical flow algorithm.

This section covers the collaboration of these techniques to create a complete object tracker. The principal idea is to track many (in our experiments 500 was a good compromise between speed and robustness) feature points and to build a feature model which detects the main direction of the moving object and corrects outliers.

The full algorithm can be summarized in these steps (Fig. 1 and 2):

1. A segmentation of the first frame is done with the Felzenszwalb & Huttenlocher algorithm, resulting in a set $\mathcal{S}$ of m regions.
2. The user selects n segments $S_1, \ldots, S_n \subseteq \mathcal{S}$ within the interested object ($1 \leq n \leq m$).
3. f feature points $F_1, \ldots, F_f \subseteq \mathcal{F}$ are computed within the selected segments. Consult section 4.1 for details.
4. Calculate the optical flow for all feature points with the Lucas & Kanade feature tracker (section 4.2). Also perform validity checks for new feature points and estimate wrongly detected feature points. Feature points are categorized into sets $\mathcal{F}_g$ for good feature points, $\mathcal{F}_e$ for estimated feature points and $\mathcal{F}_l$ for lost feature points ($\mathcal{F}_g, \mathcal{F}_e, \mathcal{F}_l \subseteq \mathcal{F}; F_g \cup \mathcal{F}_e \cup \mathcal{F}_l \equiv \mathcal{F}$).
5. Reconstruct lost feature points (section 4.3).
6. Segment the new frame, and with the help of good feature points $\mathcal{F}_g$, find segments which are likely to be part of the tracking object.
7. Repeat steps 4-6 for all remaining frames

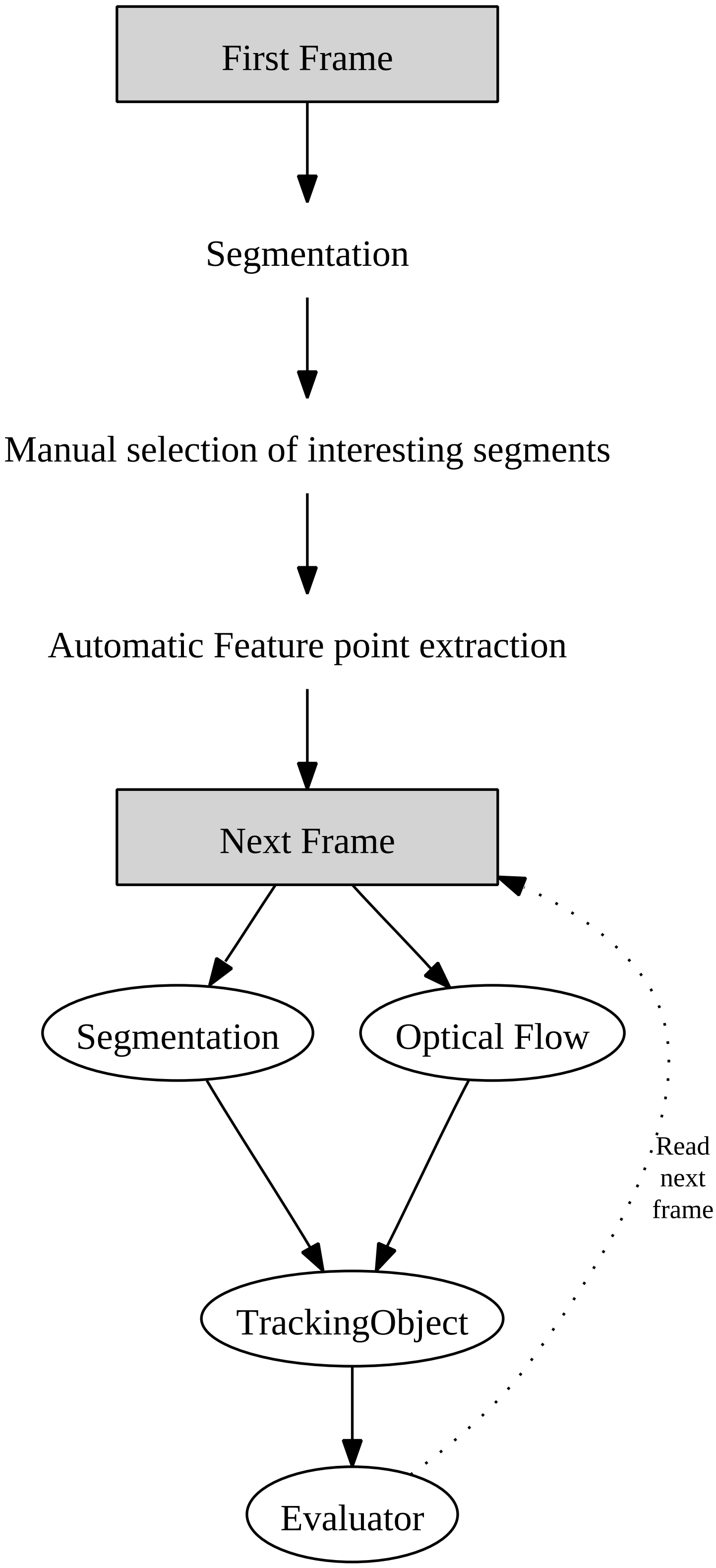
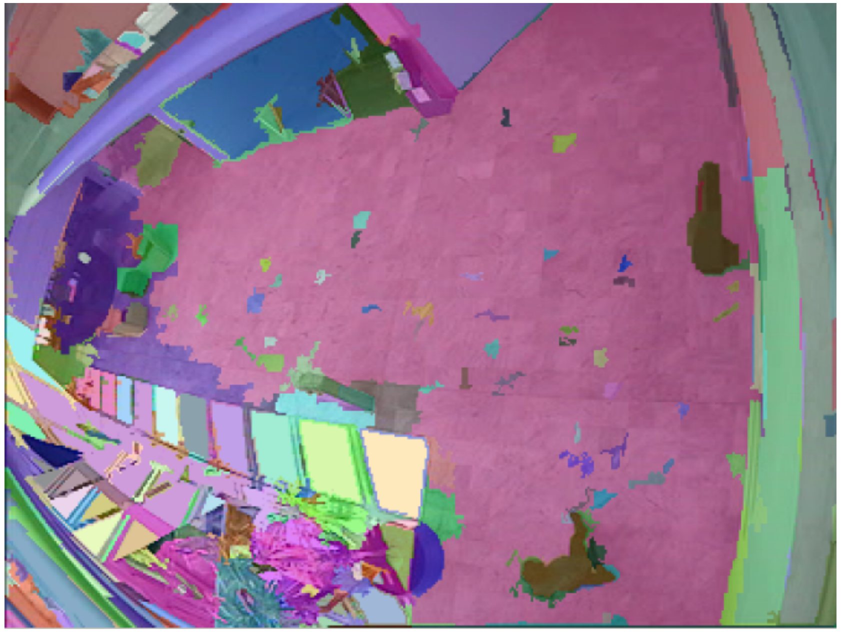


Figure 1: Workflow of algorithm

(a) The first image (from `Walk3.mpg`)



(b) Full segmentation

(c) The user selects one or more segments

(d) The selected object is being tracked (30 frames later)

Figure 2: Step by step guidance

4.1 Feature point selection

In step 3, the algorithm tries to extract good feature points, which can be easily tracked. These points have big eigenvalues in the G matrix according to [ST94]. Remember, G is the spatial gradient matrix of the image derivatives I_x and I_y. Unfortunately, the segmentation process selects regions which are usually quite uniform, and therefore good feature points are rare inside the region. Let us assume we found x good feature points, then we choose $max_F - x$ additional feature points randomly inside the region. max_F is the maximum number of feature points, in our experiments this was set to 500.

Also note that we need to avoid choosing points which lie at the border between the segment and the background. These points would be too likely to be tracked on the background in the next frame, therefore we want feature points which clearly are located within the object and not at the border. We ensure this by eroding the active segmentation with a 3×3 cross-style structuring element. This removes pixels from the border, and we can use this eroded mask to constrain new feature point positions.

4.2 Feature point calculation

Before doing any new feature point calculation, we calculate the estimated new position $P_{new}(i)$ of feature point F_i with $P_{new} = P_{old} + med(k)$ where $med(k)$ is the medium direction of the last k frames:

$$med(k) = \frac{\sum_{t=old}^{t=old-k} P_t - P_{t-1}}{k}$$

In our experiments it is shown that $k = 2$ (so just the direction of the last 2 frames) was enough, as higher values for k cannot cope with fast direction changes. Also more complex estimation formulas like using weighting newer frames with a higher factor were tried, but without better results. As it seems, the estimation of new feature positions is not that important after all, since the pyramidal implementation of the optical flow algorithm can cope with larger motions anyway.

All *good* feature point positions of $\mathcal{F}_g$ at frame position t_{old} are fed into the Lucas & Kanade feature tracker, and we get new positions of those good feature points of the last frame.

For an *estimated* feature point $\mathcal{F}_e$, we try to find a frame in the last e frames where this feature point was marked as good. Then we calculate the optical flow vector from this old frame to the current. This turned out to be a highly effective way to handle occlusion of the feature point. Setting e to higher values has the advantage that long periods of occlusion can be handled well, but has a much higher computational cost. In our experiments we set e to 20 as a trade-off between speed and robustness, but this value can be adjusted for special needs.

For all feature points of $\mathcal{F}_g$ and $\mathcal{F}_e$ we get new feature point positions with our optical flow calculations. Unfortunately, not all of these new positions are usable, therefore we need to reject some new feature points:

- Each feature point has an associated error value which is basically the color difference of the old and the new pixel. If this error is over a certain threshold τ_F, it is rejected.

- The median direction (direction vectors are rounded to the nearest integer for the median value) for all remaining good points is calculated. All points which differ more than τ_M percent from the median direction are rejected. Larger τ_M values can cope better with non-rigid objects, but are more affected by optical flow errors. We chose 20% as our threshold, but again, this depends on the actual application.

Each feature point F_i has a counter attached, which is incremented each time the point is not found or rejected in the new frame. If F_i was found however, the counter is reset to 0. Depending on the value of this counter, F_i can be:

- Moved to set $\mathcal{F}_g$ if it is reset to 0
- Moved to the set of estimated points $\mathcal{F}_e$, if it reached 1
- Moved to the set of lost points $\mathcal{F}_l$, if it reached e. Therefore we try to find an estimated point maximum e frames, otherwise we need to create a new feature point which can hopefully be better tracked (section 4.3).

4.3 Feature point restoration

Whenever feature points are declared lost, they are not immediately restored, but only when $|\mathcal{F}_l| > \frac{|\mathcal{F}|}{l}$. $l > 0$ is a factor which defines how many points must be lost before reconstructing feature points. $l = 4$ (which was our experimental setting) means, the full restoration process is started when 25% of all all feature points are lost.

The algorithm for finding suitable feature points is exactly the same as in the original feature point selection (Section 4.1).

5 Evaluation

The implementation and evaluation of the system was done on Linux in C++, using the excellent OpenCV[1] image processing toolkit. On a 2.8Ghz Intel Pentium 4, it usually ran with 1-2 frames per second, depending on frame size, number of features and the number of look-back frames e. Some example videos which show the capabilities (and problems) of our tracker can be downloaded from: http://www.prip.tuwien.ac.at/Research/twist/software.php

5.1 Input

The input to the system is a video with a fixed camera position. However, since the tracking system uses segmentation instead of background subtraction, it can usually cope with slightly moving cameras as well.

The tracked object may also be partly or fully occluded for some time, although currently just for e frames. If it is longer occluded, it is lost and cannot be automatically detected anymore.

Preferably, the object should be as rigid as possible, because otherwise different parts of an object may have completely different optical flows. Since we combine segmentation with optical flow information, it is often also possible to track non-rigid objects like people accurately.

[1] http://www.intel.com/technology/computing/opencv/

5.2 Output

After we have found a segmentation of the tracked object, visualization is shown directly on the input video. The segmentation of the object is overlayed with a transparent color and a bounding box of the object is drawn around it. Moreover, also the bounding box of an available ground truth data is drawn with a different color, so the observer can quickly see the difference of the expected and our bounding box. For debugging purposes, we also draw the feature points in different colors, depending whether they belong to $\mathcal{F}_g$, $\mathcal{F}_e$ or $\mathcal{F}_l$ (Figure 3).

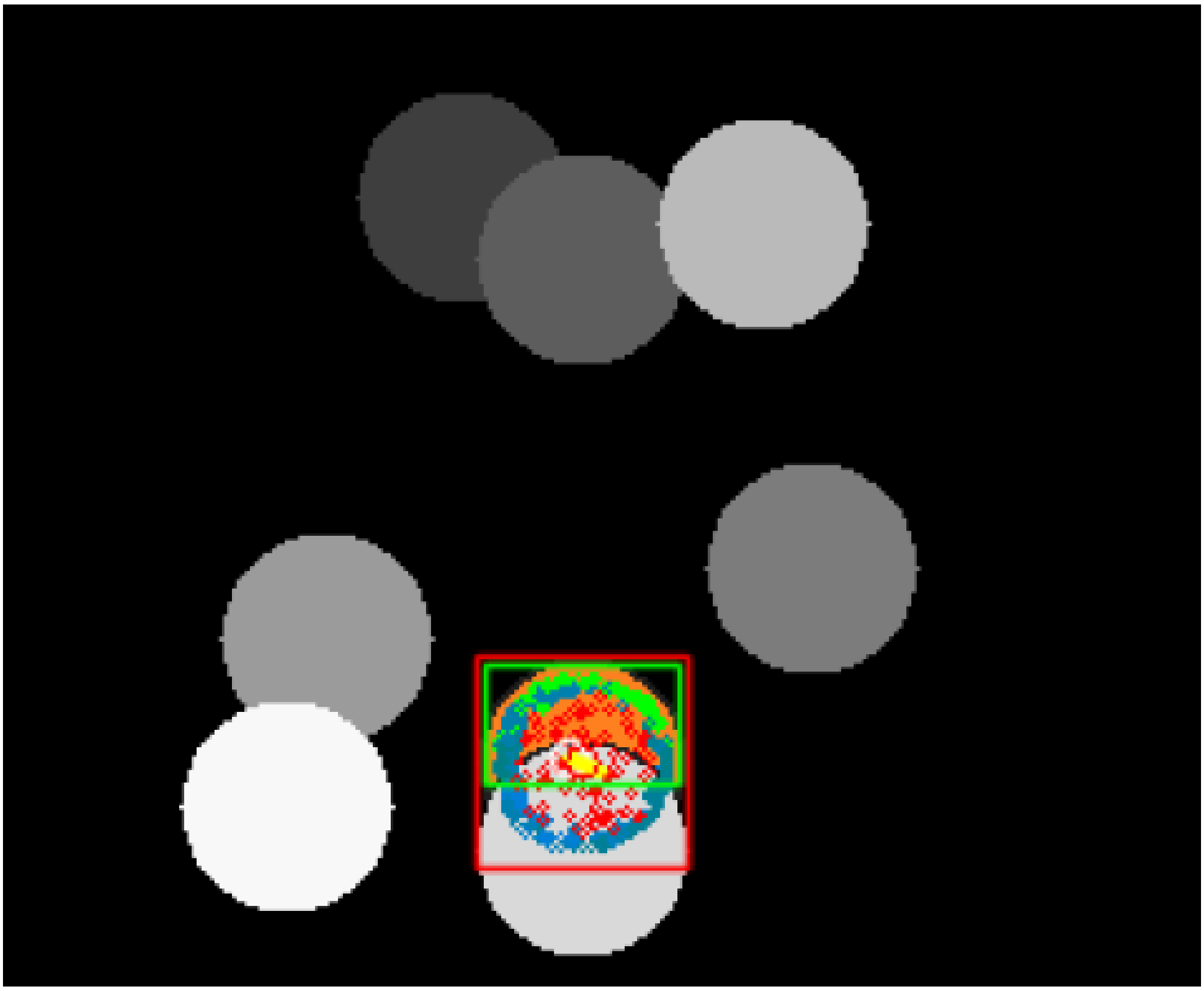


Figure 3: The tracked (partly occluded) ball in the MUSCLE benchmark. Green are found feature points, blue estimated feature points and red lost feature points. The orange color shows our segmented tracking object with its green bounding box. The red bounding box is from the ground truth data and incorrect, since it does not honor the occlusion of the object.

5.3 Parameters used

During our tests, the following parameters were used:

- Felzenszwalb-Huttenlocher segmentation algorithm:

- $\alpha = 0.5$ (Gaussian smoothing parameter)
- $k = 300$
- Minimum component size m: 20

- Lucas & Kanade optical flow algorithm:
  - window size: $\omega_x = \omega_y = 5$
  - pyramidal implementation: regular pyramid, 4 levels
- Tracking algorithm:
  - Threholds: τ_F: 100, τ_M: 20%
  - Erosion: 3×3 cross style structuring element, 3 runs
  - Number of feature points: 500
  - Look-back frames e: 20

The use of good parameters for the segmentation is essential. Sometimes, an object melted with the background, and it was impossible to perform a tracking operation. Changing α, k or m sometimes helped here, but not always.

The optical flow and tracking algorithm parameters did not really have much effect when they were altered.

5.4 Results

While overlaying our tracking and segmentation over the original video may look nice, we need to prove how good (or bad) our results are. This can only be done with a video sequence where ground truth data is available.

Freely available video sequences with annotated ground truth data are very hard to find, but the CAVIAR project has some nice real world videos with hand selected bounding boxes for each frame in XML format[2]. Credit for these videos goes to the *EC Funded CAVIAR project/IST 2001 37540*.

We also evaluated the tracker on the MUSCLE Brenchmark from http://muscle.prip.tuwien.ac.at/data_description/ACV1/ACV_TRACKINGBENCHMARK.HTML. Credit for these videos goes to the *Advanced Computer Vision GmbH - ACV*. They have ground truth annotated videos of moving circles. While segmentation is much easier for these videos because they are artificial without any noise or soft gradients, it can be quite difficult to handle occlusion for these blobs.

We compared the annotated bounding boxes of an object with our results and got the following evaluation criteria:

- **Overlap**
  This is just the comparison of overlapping bounding boxes. While more accurate calculation of overlap of the segmented region with ground truth data would be favorable, none of the ground truth data provided more information than simple bounding boxes. In the case of the MUSCLE benchmark, one could approximately calculate the circles from the bounding boxes, not only would this be inaccurate, but it wouldn't take occlusion into account either.

  Also the exact overlap values for the MUSCLE benchmark would be identical for our experiments. In the case of a circle with radius $r = 20$, but our segmentation incorrectly segments a smaller circle with $r = 18$ within the ground truth circle.

[2] The videos and the XML files for the ground truth can be downloaded from: http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1/

Exact overlap is:

$$O_c = \frac{18^2 \cdot \pi}{20^2 \cdot \pi} = 0.81$$

For the bounding boxes, the overlap is:

$$O_b = \frac{(18 \cdot 2)^2}{(20 \cdot 2)^2} = 0.81$$

- **Percentage of successfully tracked frames**
  This measurement just uses a threshold of the overlapping percentage to classify if the object was successfully tracked or not in a frame. It was chosen as low as 25%. This may sound like a very low threshold, but as we compare the overlapping area of the bounding boxes, it is very common to have 40% overlapping bounding boxes which still look well tracked. Of course this threshold can be changed, if you need more confidence for a tracked object.

In Table 1 we have summarized some results for the CAVIAR benchmark. The results vary, and objects with strong borders like in `Browse1.mpg` could be tracked very well. On the other side, the tracking object of `OneStopMoveNoEnter1cor.mpg` or `Walk2.mpg` was completely lost after some time and could not be recovered. This however, was not caused by the feature tracker, but by bad segmentation results. Often, not only the moving person but also parts of the background were segmented as one connected region. This did not just lead to bad results for the *overlap* statistics, but when we needed to reconstruct lost feature points, they were sometimes taken on the background. Therefore, for real world scenes our proposed tracker would need a better segmentation algorithm, or at least a mechanism to make sure, that reconstructed feature points are not taken on the background but within the tracking object.

Filename	**ObjectID**	**Frames**	**Succ. Tracked**	**Overlap**
`Browse1.mpg`	1	1–200	90.12%	70.26%
`OneLeaveShop1cor.mpg`	0	1–90	97.78%	85.51%
`OneStopMoveEnter2cor.mpg`	0	1–300	99.00%	54.26%
`OneStopMoveNoEnter1cor.mpg`	4	1300–1664	37.36%	30.43%
`Walk2.mpg`	0	1–50	26.00%	15.90%
`Walk3.mpg`	0	50–185	100.00%	80.21%

Table 1: Tracking performance for the CAVIAR benchmark

In the MUSCLE benchmarks, the results are better (Table 2), as the segmentation was not a real problem here. The ball could usually be fully tracked throughout the video (Fig. 3). However, the ground truth files were not that exact and contained bounding boxes of the ball even if it was occluded. Therefore the *Successfully tracked* column is not always at 100%, even if the ball is successfully tracked in each frame.

The *overlap* column shows an average overlap of about 70%. We expected higher values, since this artificial video has a high contrast between the balls and the background. Yet unfortunately, the Felzenszwalb segmentation does not segment the ball as one component but puts the contour of the ball into an own component (Fig. 4).

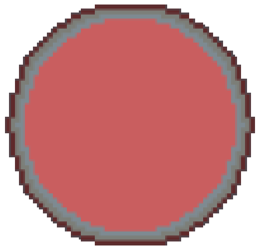


Figure 4: The Felzenszwalb algorithm yields three components for the ball, while one component would be preferable.

Filename	**ObjectID**	**Frames**	**Successfully Tracked**	**Overlap**
CASE13_00001.AVI	2	2–1200	100.00%	80.06%
CASE14_00002.AVI	3	2–510	99.41%	73.47%
CASE14_00003.AVI	7	2–650	93.07%	67.79%
CASE16_00004.AVI	13	2–1357	98.49%	59.89%
CASE16_00005.AVI	44	2–1000	99.90%	62.01%
CASE16_00005.AVI	45	2–450	100.00%	61.70%

Table 2: Tracking performance for the MUSCLE benchmark

6 Outlook and Conclusion

We could see that combining an optical flow based feature tracker with a segmentation can lead to a usable object tracker. The most important aspect of this work was building and maintaining the feature model. This was done with three sets for found, estimated and lost feature points. Using this model, also dealing with occlusion was quite successful.

However, there are still some problems and improvements, which are beyond the scope of this work. The Felzenszwalb and Huttenlocher segmentation algorithm does not really work well if there is no strong border between object. And even if there is a strong border like in the MUSCLE benchmark, the segmentation results could be better. However, on the other side, it could serve quite a usable segmentation for many input videos, so there must be some proof that other segmentation algorithms really work that much better.

Another field for improvement is the feature restoration process. It works well as long as the segmentation process does not segment the background as part of the object.

A third improvement would be to eliminate the manual process of initially selecting a tracking object. In the current implementation this is done by clicking into the desired object, future implementations could use background subtraction or other techniques to make the tracker fully automated.

Apart from these problems, the tracker works reasonably well and even handles deformations and rotation of the object to a certain degree.

References

[AB85] Edward H. Adelson and James R. Bergen. Spatiotemporal Energy Models for the Perception of Motion. *J. of the Optical Society of America A*, 2(2):284–299, 1985.

[Bou03] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the Algorithm. Part of OpenCV Documentation, 2003.

[CSE05] A. Cavallaro, O. Steiger, and T. Ebrahimi. Tracking Video Objects in Cluttered Background. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(4):575–584, 2005.

[FH04] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Graph-Based Image Segmentation. 59(2):167–181, 2004.

[HK03] Yll Haxhimusa and Walter G. Kropatsch. Hierarchy of Partitions with Dual Graph Contraction. volume 2781, pages 338–345, Germany, 2003. Springer.

[HK04] Yll Haxhimusa and Walter G. Kropatsch. Segmentation Graph Hierarchies. In Ana Fred, Terry Caelli, Robert P.W. Duin, Aurelio Campilho, and Dick de Ridder, editors, *Proceedings of Joint International Workshops on Structural, Syntactic, and Statistical Pattern Recognition S+SSPR 2004*, volume 3138, pages 343–351, Lisbon, Portugal, 2004. Springer, Berlin Heidelberg, New York.

[HS80] Berthold K.P. Horn and Brian G. Schunck. Determining Optical Flow. Technical report, MIT, Cambridge, MA, USA, 1980.

[JR93] Jean-Michel Jolion and Azriel Rosenfeld. *A Pyramid Framework for Early Vision: Multiresolutional Computer Vision*. Springer, 1993.

[LK81] B.D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *IJCAI81*, pages 674–679, 1981.

[MMTB$^+$06] R. Marfil, L. Molina-Tanco, A. Bandera, J. A. Rodriguez, and F. Sandoval. Pyramid Segmentation Algorithms Revisited. *Pattern Recognition*, 39(8):1430–1451, August 2006.

[Ros00] Michael G. Ross. Exploiting Texture-Motion Duality in Optical Flow and Image Segmentation, April 2000.

[SA91] E. Simoncelli and E. Adelson. Computing Optical Flow Distributions using Spatio-Temporal Filters, 1991.

[SKK$^+$05] Jeongho Shin, Sangjin Kim, Sangkyu Kang, Seong-Won Lee, Joonki Paik, Besma Abidi, and Mongi Abidi. Optical Flow-Based Real-Time Object Tracking using Non-Prior Training Active Feature Model. *ELSEVIER Real-Time Imaging*, 11:204–218, June 2005.

[ST94] Jianbo Shi and Carlo Tomasi. Good Features to Track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994.

[WDD00] Yiwei Wang, John F. Doherty, and Robert E. Van Dyck. Moving Object Tracking in Video. In *AIPR '00: Proceedings of the 29th Applied Imagery Pattern Recognition Workshop*, page 95, Washington, DC, USA, 2000. IEEE Computer Society.