SPARSE OPTICAL FLOW REGULARIZATION FOR REAL-TIME VISUAL TRACKING

Vincent Spruyt[1,2], *Alessandro Ledda*[1] *and Wilfried Philips*[2]

[1]Dept. of Applied Engineering: Electronics-ICT,
Artesis University College Antwerp
Paardenmarkt 92, 2000 Antwerpen, Belgium

[2]Ghent University-TELIN-IPI-IMINDS
St. Pietersnieuwstraat 41, 9000 Gent, Belgium
v.spruyt@ieee.org

ABSTRACT

Optical flow can greatly improve the robustness of visual tracking algorithms. While dense optical flow algorithms have various applications, they can not be used for real-time solutions without resorting to GPU calculations. Furthermore, most optical flow algorithms fail in challenging lighting environments due to the violation of the brightness constraint. We propose a simple but effective iterative regularisation scheme for real-time, sparse optical flow algorithms, that is shown to be robust to sudden illumination changes and can handle large displacements. The algorithm proves to outperform well known techniques in real life video sequences, while being much faster to calculate. Our solution increases the robustness of a real-time particle filter based tracking application, consuming only a fraction of the available CPU power. Furthermore, a new and realistic optical flow dataset with annotated ground truth is created and made freely available for research purposes.

Index Terms— Optical Flow, Particle filter, Object Tracking

1. INTRODUCTION

Optical flow is defined as a 2D vector field describing the apparent velocities of objects in a 3D scene, projected onto the image plane. Most techniques to calculate such flow fields rely on so called brightness constancy, assuming that the intensity of a small region remains constant between two frames, despite its movement.

A widely used algorithm based on this assumption is the pyramidal Lucas-Kanade optical flow algorithm which tries to analytically solve the optical flow equations simultaneously for all pixels within a small region. Due to the well known aperture problem, the Lucas-Kanade algorithm is mostly used to calculate optical flow only at corner points, where the flow equations are well defined. The resulting flow field is then sparse, and algorithms computing such flow field are called *sparse optical flow algorithms*.

To overcome the aperture problem, variational techniques which introduce a smoothness constraint have been introduced. The well known Horn and Schunck method defines a global smoothness constraint which effectively regularizes the flow field, allowing the calculation of flow vectors at each image point. These methods are thus called *dense optical flow algorithms*. The main disadvantage of their approach however, is that flow vectors at motion discontinuities are unstable because of the global regularization term.

Recently, several interesting results have been obtained using variational techniques such as the Motion-Detail-Preserving-Flow (MDP-Flow) algorithm proposed by Li Xu *et al.* [1]. Their implementation is currently top-ranked at the widely referred Middlebury optical flow evaluation [2] and is able to maintain fine object boundaries while calculating a dense flow field. Dense optical flow algorithms however, usually need seconds up to minutes of calculation per video frame on current hardware, before obtaining the resulting flow field and are therefore not suited for real-time applications.

Research funded by a PhD grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

Furthermore, several important real-time applications such as object tracking do not directly benefit from a dense optical flow calculation. In these applications, optical flow calculations should only consume a small portion of the available computational resources and is allowed to be sparse.

Because of their real-time constraints, these applications often fall back to the Lucas-Kanade optical flow algorithm as calculation speed of the obtained flow vectors is more important than the density and correctness of the flow field. Kodama *et al.* [3] use a Lucas-Kanade based optical flow estimate to enhance the state transition model in a probabilistic tracking framework. Lucena *et al.* [4] enhance the observation model of a particle filter by incorporating the Lucas-Kanade flow. Kristan *et al.* [5] obtain Lucas-Kanade estimates to locally adapt the observation model for each tracked object.

A major disadvantage of this approach, however, is that the resulting flow field will be of poor quality in case of sudden illumination changes while exactly at those moments, the robustness of the obtained flow field is of great importance to the object tracker when trying to re-initialize its observation models. Also, sparse optical flow algorithms do not incorporate a regularization term and are thus more susceptible to noise than dense optical flow solutions.

Schikora *et al.* [6] employ a more advanced and robust dense optical flow algorithm based on Total Variation to improve a finit set theory based particle filter. They achieve near real-time performance by implementing the optical flow algorithm on the GPU. However, many other, more important algorithms used in a complete tracking solution, such as particle filtering, active contour optimization and segmentation, can benefit from GPU implementation. Therefore one has to carefully select the tracking phase to devote the GPU to.

In this paper, we propose a real-time, sparse optical flow algorithm which is robust to sudden illumination changes and large displacements, and consumes only a fraction of the processor time without resorting to GPU calculations. Furthermore, a local regularization term is included in the optimization function, yielding a smooth flow field while preserving motion boundaries. The algorithm runs faster than the pyramidal Lucas-Kanade algorithm, and outperforms more advanced algorithms in accuracy. The proposed solution can therefore replace sparse optical flow algorithms used in real-time applications in order to improve their robustness without negatively impacting their performance.

2. FEATURE DETECTION AND DESCRIPTION

In order to avoid the aperture problem, sparse optical flow is only calculated for corners, which are regions containing both large horizontal and large vertical gradients. A widely used corner detector is the Harris corner detector which relies on the magnitude of the eigenvalues of a structure tensor matrix describing the local change in brightness. This solution however is rather slow and consumes a large portion of the resources available to the optical flow algorithm.

E. Rosten *et al.* introduced a machine learning based corner detector named FAST [7]. The learned decision tree yielded an extremely fast corner detector. Furthermore, the authors showed that their detector has great repeatability under various aspect changes.

A disadvantage of the FAST detector, however, is its reliance on a threshold which eliminates corners with a low score. In the original implementation, a fixed single threshold is used. However, this approach means that in certain lighting conditions more corners will be found than in others. Simply dynamically adapting the threshold in order to maintain an approximately constant number of corner detections in subsequent frames, does not completely solve this problem because using a single threshold will then return corners on highly textured areas, while completely ignoring less textured regions.

We propose a simple and efficient change to the original FAST detector allowing it to detect corners that are more uniformly distributed over the image while automatically adapting to changing environments and illumination conditions. Instead of defining a single threshold, the image is divided into square regions of size $N \times N$, and a threshold is calculated for each of these regions. Once such threshold is obtained, bilinear interpolation is used to determine a threshold for each pixel location in the image. To obtain a threshold for each $N \times N$ block, a simple iterative procedure is used. Initially, the threshold is set to the threshold used in the previous frame. Next, corners are detected after which the thresholds are decremented if less than a certain number of corners is found in the block, while they are incremented if more than a maximally allowable number of corners is found. The minimum threshold is then limited in order to avoid detecting noisy corners in untextured regions of constant brightness. This procedure is repeated until the desired number of corners is detected or a maximum number of iterations has passed.

This procedure ensures that approximately the same number of corners is found in subsequent video frames, even under extreme lighting changes. For each corner, an illumination invariant descriptor is then built which allows it to be matched to a corner in the previous frame. A well known illumination independent primitive used in texture classification, are Local Binary Patterns (LBP) [8], in which the sign of the difference between a center pixel and each of its neighboring pixels in a 3×3 neighborhood is multiplied with a binomial factor in order to obtain a binary number between 0 and $2^8 - 1 = 255$. This number then represents the texture in that area. Since only simple addition and subtraction is needed and multiplication by a power of two which can be implemented efficiently using shifting operators, calculating an LBP is extremely fast. The histogram of the LBPs computed over a region can then be used as a texture descriptor.

However, the classical LBP operator yields very long histograms of 256 bins, many of which would be empty. Furthermore, comparing two such histograms would be too slow for our application. M. Heikkilä *et al.* [9] defined a modified operator which they call the Center-Symmetric LBP (CS-LBP), in which the sign of the difference between center-symmetric pairs of pixels is used instead of the difference between each pixel and the central pixel, as illustrated by (1). In a 3×3 neighborhood this results in a CS-LBP number between 0 and $2^4 - 1 = 15$ which greatly reduces the number of histogram bins. The authors showed that CS-LBP based descriptors exhibit superior performance when compared to classical LBP.

$$CS\text{-}LBP(x, y) = \sum_{i=0}^{3} 2^i \, \mathbf{1}_{\mathbb{Z}+}(n_i - n_{i+4}) \qquad (1)$$

where $\mathbf{1}_{\mathbb{Z}+}$ is an indicator function. In this paper, CS-LBP descriptors are calculated for a 18×18 region centered on a detected corner. In order to incorporate spatial information into the descriptor, this region is divided into 9 cells of size 6×6, and a CS-LBP histogram is calculated for each cell. The histograms are then concatenated to obtain the final spatially enhanced CS-LBP based feature vector of size $16 \times 9 = 144$.

3. SPATIAL CLUSTERING

We propose a local regularization scheme that can be used to regularize sparse optical flow algorithms while still being able to cope with discontinuities. Although we use FAST corners and CS-LBP histograms to obtain the initial sparse flow field in real-time, our regularization scheme can be combined with other, more advanced feature detectors and descriptors such as SURF.

Global regularization of the optical flow field, such as used by the Horn and Schunck method, would assume a globally smooth vector field and thus cannot handle motion discontinuities at object boundaries. Instead, the optical flow field should only be regularized locally. Therefore, a method is needed to define a local neighborhood for each detected corner in an illumination invariant manner. The solution should have low complexity and should resemble a rough segmentation of the objects in an image.

Instead of segmenting the image based on its lighting dependent color distribution, we propose to segment it based on its corner distribution. This means that image regions containing high spatial frequencies such as textures will be separated from regions containing only low spatial frequencies such as smoothly varying background.

An efficient method to obtain such a segmentation is to spatially cluster the detected corner points. However, corner locations often exhibit clear patterns and can be almost co-linear if they are part of an edge, which means that fast traditional clustering algorithms such as k-means clustering, which tries to find spherical clusters, are not suited for this goal. Instead, a Minimum Spanning Tree (MST) based clustering is used as such solutions are known to be capable of detecting clusters with irregular boundaries [10] and do not need to know the number of clusters in advance.

In order to efficiently determine the MST, a Delaunay triangulation is calculated for the detected corners which can be implemented very efficiently for Euclidean point sets by using a radial sweep-hull scan [11]. From the resulting triangulation, when considered as a Euclidean graph, the minimum spanning tree is obtained by means of the Kruskal algorithm.

To obtain a clustering, inconsistent edges are removed from the MST. An edge is regarded inconsistent if one of following conditions holds:

$$\begin{cases} \|N_1 - N_2\| > \mu_{N_1} + c\,\sigma_{N_1} \\ \|N_1 - N_2\| > \mu_{N_2} + c\,\sigma_{N_2} \end{cases} \qquad (2)$$

where $\|N_1 - N_2\|$ is de Euclidean length of the edge and μ_{N_1} and μ_{N_2} are respectively the average length of nearby edges at the side of node $N1$ and the average length at the side of node $N2$. σ_{N_1} and σ_{N_2} are the standard deviations of the edge lengths at both sides of the edge $N_1 N_2$. In our implementation, c was set to $c = 1.5$, and two nodes are considered '*nearby*' if their geodesic distance is not larger than three. Figure 1 illustrates the result of this clustering approach on the detected corners of a scene in two video frames. In the second video frame, illumination has changed and objects have moved compared to the first frame.

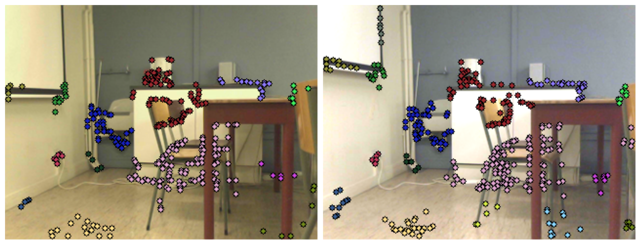


Fig. 1. Spatial corner clustering in subsequent video frames

4. FEATURE MATCHING

In order to obtain the optical flow field, corners from the current frame are matched with corners from the previous frame, based on their descriptors. As discussed in section 2, each corner is described by a 144 bin histogram. Thus, finding correspondences simply means searching for a corresponding corner such that a cost function between both corner descriptors, LBP_1 and LBP_0, is minimized. Well known distance measures for comparing histograms are the Chi-Square distance or the Bhattacharyya distance. However, for performance reasons, we chose to use the L_1 norm which yields similar results while being much faster to calculate.

Matches from a corner c_{1i} in the current frame to a corner c_{0j} in the previous frame are searched for within a certain radius r surrounding c_{1i}. In this paper, the radius was set to $r = 30$, using a frame resolution of 320×240, thereby allowing large displacements between subsequent frames. However, while such large displacements are allowed, they are highly unlikely to occur in real video. Assuming this likelihood drops linearly with the displacement distance, the Euclidean distance between corner c_{1i} and a candidate corner c_{0j} is added to the cost function to be minimized. The addition of this term makes sure that large displacements are penalized more than small flow vectors.

Finally, a regularization term $\lambda(i)$ is added to steer the search process such that the obtained flow field is locally smooth. While corner matching itself does not take the previously defined clusters into account, the regularization term does, as will be discussed in the following section. The cost function is thus defined as follows:

$$C(c_{1i}, c_{0j}) = |LBP_1 - LBP_0| + \alpha \|c_{1i} - c_{0j}\| + \beta\lambda(i) \tag{3}$$

where α and β are weighting factors indicating the importance of the regularization terms. In our experiments, $\alpha = 1$ and $\beta = 100$.

Initially, the regularization term is defined $\lambda(i) = 0$. The matching process is then an iterative process, where in the first iteration the non-regularized optical flow is obtained. Based on this flow field, the regularization term for the next iteration is calculated, and the process is repeated until convergence or the maximum number of iterations has been reached.

5. REGULARIZATION TERM

To enforce local smoothness, local regularization is based on weighted vector median filtering. Vector median filters are known to be able to smooth estimated velocity fields while maintaining motion discontinuities or edges. L. Alparone *et al.* [12] showed that the application of a weighted vector median filter after dense optical flow calculation improves the smoothness and accuracy of the result. The disadvantage of such approach however, is that the median filter is only used as a non-linear postprocessing filter to reduce noise in a non-regularized flow field instead of incorporating the smoothness constraint directly into the search process itself.

Furthermore, while it is clear how to apply a filter on a dense optical flow field on a regular grid, it is less obvious how to use a similar technique on a sparse flow field in a non-regular grid. This section discusses how vector weighted median filtering can efficiently be used on a non-regular grid to directly incorporate the smoothness constraint as a regularization term in the cost function.

The vector median filter is basically just a median filter applied to each component of the vectors. In order to regularize the optical flow search process, we define the optimal flow vector for a corner c_{1i} as the vector obtained in that point after weighted vector median filtering of the flow field obtained in a previous iteration, and then minimize the difference between a candidate flow vector and the optimal flow vector by means of a well chosen regularization term λ.

In order to obtain the median vector within a neighborhood of a corner c_{1i}, the concept of neighborhood must be defined for a sparse, non-regular grid. We propose to adopt the concept of natural neighbors as used in Voronoi interpolation schemes. The natural neighbors of a corner c_{1i} are those corners whose Voronoi cell in a Voronoi tessellation are adjacent to the Voronoi cell of corner c_{1i}. Furthermore, only corners belonging to the same cluster as corner c_{1i} are considered which effectively allows us to perform regularization locally and to deal with motion discontinuities. Calculating the Voronoi tessellation incurs almost no extra cost, since a Delaunay triangulation was obtained already in the clustering stage, and the Voronoi diagram is nothing more than the dual graph of this Delaunay triangulation.

For each corner c_{1i}, the weighted median flow vector can then be calculated based on its valid neighbors, which are those neighboring corners that are natural neighbors and belong to the same cluster as corner c_{1i}. A weighted median filter replicates each element several times, based on its weight, before finding the median value. The weighting factor for a certain vector should depend on both the reliability of the estimated flow vector in the previous iteration, and a distance measure indicating how close the neighbor is to corner c_{1i}.

The reliability of the estimated flow vector is obtained directly from the cost $C(c_{1i}, c_{0j})$ associated with the match. For the distance measure, we propose to use the weighting factor as used in the non-Sibsonian Voronoi interpolant for non-regular grids, described by Belikov *et al.* [13], as this weighting factor effectively uses the natural neighbor concept by defining the weight as the quotient of the distance between the centroids of the cells $\|H_{kl}\|$ and the length of the edge S_{kl} that is shared by the two Voronoi cells. This is illustrated more clearly in figure 2.

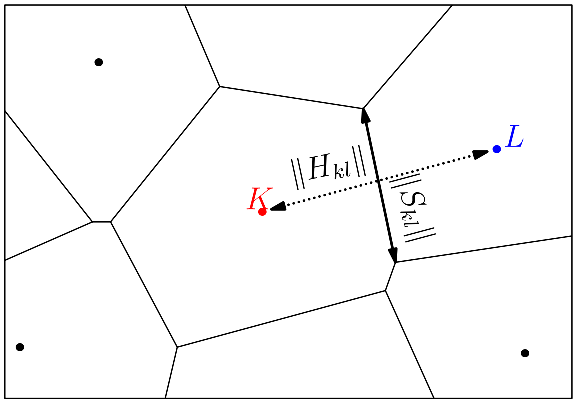


Fig. 2. Definitions for the non-Sibsonian interpolant weight

For each neighboring corner l, the weight w_l, used by the weighted vector median algorithm, is then defined by:

$$w_l = C(c_{1i}, c_{0j}) \frac{\|H_{kl}\|}{\|S_{kl}\|} \tag{4}$$

Each neighboring element is then replicated R_l times, after which the median is found in the resulting vector of elements.

$$R_l = \frac{\max(\mathbf{w}) + 1}{w_l + 1} \tag{5}$$

The optimal flow vector $\mathbf{o_i}$ for a corner point c_{1i} is the vector obtained after weighted median filtering the u and v components of the neighbors of this corner. In the next iteration, the difference between a possible candidate vector and this optimal vector is minimized in order to obtain a smooth flow field. Minimization of this difference is achieved by defining a regularization term $\lambda(i)$ in the cost function as follows:

$$\lambda(i) = |\,\|\mathbf{o_i}\| - \|\mathbf{f_i}\|\,| \arccos\left[\frac{\langle \mathbf{o_i}, \mathbf{f_i}\rangle}{\|\mathbf{o_i}\|\|\mathbf{f_i}\|}\right]^2 \tag{6}$$

where $\mathbf{f_i}$ is a candidate flow vector.

Adding the regularization term to the cost function makes sure that the difference in magnitude between the optimal vector $\mathbf{o_i}$ and the candidate vector $\mathbf{f_i}$ is minimized, together with the squared angle between both vectors. By squaring the angle, a change in direction is penalized more than a change in length.

The final flow field obtained is either the result of the last iteration, or the result of the last median calculation. In the latter case, which is what was used in our experiments, this corresponds to applying a final weighted median filter as a postprocessing step to the already regularized optical flow field.

6. EVALUATION

Evaluating optical flow algorithms is a tedious and difficult task, mainly because of the lack of ground truth data. The most widely used dataset of ground truth optical flow vectors is the Middlebury optical flow database [2]. However, the Middlebury dataset was created for typical optical flow algorithms based on the constant brightness assumption and therefore does not contain sequences with changing illumination. Furthermore, the available test sequences are synthetically generated and often do not resemble real video very well as the video frames contain very fine grained details, a lot of similarity in texture, and frame to frame displacements are rather small. Also the sequences lack motion blur and realistic illumination, resulting in unfair comparison between algorithms that can deal with noisy observations, and algorithms that assume perfect data.

This led us to the creation of a real-video dataset which is made publicly available together with its carefully annotated ground truth[1]. However, even though the algorithm described in this paper was designed with changing lighting conditions and fast motion in mind, it might still be interesting to see how it performs on artificial data. Therefore, in the next sections, the algorithm is evaluated both on the Middlebury dataset, and on a dataset with changing illumination and fast motion for which ground truth data was manually created.

In our experiments, the image size was 320×240, the search radius was 30 and three iterations were used for regularization.

For each sequence in the dataset, optical flow fields are calculated by our proposed algorithm, indicated as *LBP-flow*, by the pyramidal *Lucas- Kanade* algorithm, by the Total Variation based *MDP-flow* algorithm [1], and by *SURF* descriptor based matching. While Lucas-Kanade represents the most widely used motion estimation technique in real-time object tracking, MDP-flow is currently the top-ranking algorithm when evaluated on the Middlebury dataset. SURF descriptor based matching is often used for point tracking and is considered one of the most robust descriptors currently available.

For the pyramidal Lucas Kanade algorithm, a window size of 10×10 and a 2-level pyramid was used, resulting in a maximally allowed displacement of $(2^2 - 1) \times 10 = 30$ pixels which is the same as the search radius used by our algorithm and the SURF algorithm. Changing these parameters did not affect the results much. For the SURF algorithm, it is important to note that only SURF descriptors were used to describe and match the FAST feature points, and that the SURF detector itself was not used. The reason for this is that the SURF detector returns Determinant of Hessian (DoH) based features which can not be fairly compared with corners.

Figure 3 shows the resulting angular deviation as described in [2] from the given ground truths for each of the algorithms tested on the Middlebury sequences. Figure 4 shows the resulting endpoint deviation for these same sequences and algorithms. Endpoint errors simply correspond to the difference in vector length between the estimated flow and the given ground truth. These results indicate that the proposed algorithm, *LBP-flow*, outperforms both Lucas-Kanade and the SURF matching algorithm. For each of the test sequences, except for the *Grove2* sequence, a lower average angular error is obtained, accompanied by a lower endpoint-error. While SURF descriptors are much more descriptive than our CS-LBP descriptor and are rotation invariant, our proposed regularization scheme allows the use of much simpler feature descriptors resulting in faster execution.

The results also show that the offline *MDF-flow* algorithm seems to be much more robust than the proposed real-time algorithm. However, it is important to note that the Middlebury dataset does not try to mimic real-life, low-quality video sequences. Real video data tends to lack sharp textures and edges, contains large and irregular displacements, and a minimal amount of spatial self-similarity. Therefore, algorithms performing well on this dataset might not perform well on real video data and vice versa.

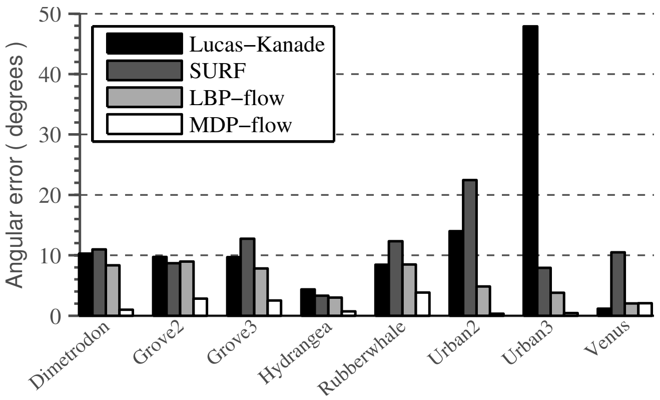


Fig. 3. Middlebury Dataset evaluation (angular error)

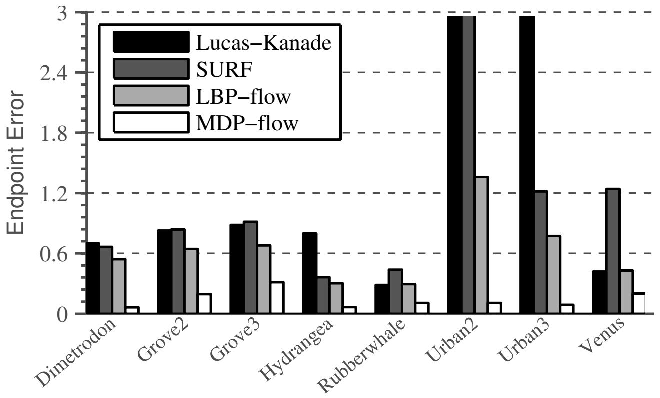


Fig. 4. Middlebury Dataset evaluation (endpoint error)

Table 1 shows the angular errors and table 2 shows the endpoint errors obtained by each algorithm on the real-life video sequences we created and made public for the research community.

These sequences contain large and irregular displacements up to 30 pixels, webcam artifacts such as vignetting, noise and motion blur, and sudden changes in lighting conditions.

Table 1. Proposed Dataset evaluation (angular error)

Sequence	Algorithm	Angular error		
		Median	Avg.	Std. Dev.
Desk	**LBP-Flow**	**1.9**	**2.0**	**2.9**
	Lucas-Kanade	87.6	87.9	65.0
	MDP-Flow	12.9	44.6	48.8
	SURF	2.0	2.5	3.0
Drinks	**LBP-Flow**	**1.1**	**3.8**	**4.2**
	Lucas-Kanade	84.7	75.1	62.0
	MDP-Flow	79.7	91.4	75.2
	SURF	7.4	15.9	31.5
Office	**LBP-Flow**	**3.3**	**3.5**	**2.3**
	Lucas-Kanade	7.2	41.5	54.5
	MDP-Flow	11.2	16.2	17.5
	SURF	8.7	27.4	41.8
Classroom	**LBP-Flow**	**5.2**	**4.8**	**3.9**
	Lucas-Kanade	6.6	21.5	37.5
	MDP-Flow	8.0	23.1	38.8
	SURF	5.4	22.0	43.4

[1]http://telin.ugent.be/ vspruyt/ICME2013/

Table 2. Proposed Dataset evaluation (endpoint error)

Sequence	Algorithm	Endpoint error		
		Median	Avg.	Std. Dev.
Desk	**LBP-Flow**	**1.0**	**2.2**	**3.0**
	Lucas-Kanade	46.2	44.4	27.0
	MDP-Flow	26.0	32.0	26.8
	SURF	2.1	2.8	3.3
Drinks	**LBP-Flow**	**1.4**	**1.8**	**1.5**
	Lucas-Kanade	18.4	23.3	25.3
	MDP-Flow	17.9	32.0	29.6
	SURF	4.0	12.0	26.1
Office	**LBP-Flow**	**1.0**	**1.1**	**0.6**
	Lucas-Kanade	2.2	8.6	10.1
	MDP-Flow	3.9	5.5	4.3
	SURF	2.0	8.3	15.8
Classroom	**LBP-Flow**	**1.4**	**1.3**	**0.7**
	Lucas-Kanade	1.8	5.1	11.1
	MDP-Flow	3.5	7.7	11.7
	SURF	1.4	10.9	22.8

The results illustrate the robustness of the proposed approach for real-life video. The bad performance of Lucas-Kanade is easily explained by the violation of the brightness constraint. While the *MDP-Flow* algorithm performs reasonably well when compared to the Lucas-Kanade algorithm, it also fails when lighting conditions change. The best result for the *MDP-Flow* was obtained with the *Office* sequence, which contains the least amount of lighting change.

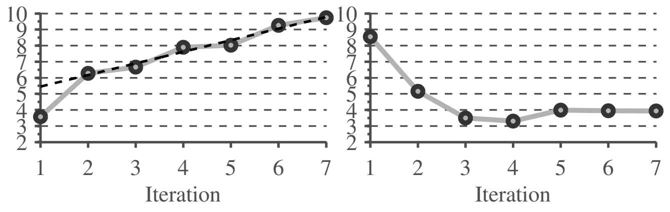
For these challenging sequences, which represent typical scenarios in object tracking applications, point matching algorithms such as the SURF method or the proposed CS-LBP based method, outperform most optical flow approaches. Furthermore, these results clearly show the advantage of our regularization scheme in combination with a simple feature descriptor as opposed to a more advanced feature descriptor without regularization constrains.

Table 3 lists the average execution times for each algorithm when the flow is calculated for 300 positions. Timings were measured on an Intel 2.4 Ghz dual core CPU. The proposed algorithm is extremely fast thanks to the simple LBP based feature descriptors used, and can thus be used in real-time tracking applications.

Table 3. Average execution time per video frame

Algorithm	Execution time (ms)	# corners
LBP-Flow	**6.6**	300
Lucas-Kanade	8.4	300
MDP-Flow	618.4	300
SURF	20.3	300

Figure 5 shows both the average execution time and the average angular error as a function of the number of regularization iterations. As indicated by the dotted line, the complexity of the proposed regularization scheme is linear in the number of iterations. Furthermore, the error clearly reduces after the first few iterations, illustrating the benefit of local regularization.



(a) Execution time (ms) (b) Angular error (degrees)

Fig. 5. Execution times and error measurements

If only a single iteration is used, no regularization is performed, and the angular error would be larger than the error obtained by more advanced point matching algorithms such as SURF. However, after a few regularization iterations, the error becomes much lower than the error obtained by other, non-regularized, methods. Generally, the algorithm converges after 3 or 4 iterations which corresponds to an execution time of about 7 milliseconds on our current hardware. The small increase in error after the fourth iteration can be explained by imperfect clustering, causing local over-smoothing of the flow field.

Figure 6 shows the video frames and corresponding flow field for each of the sequences used for evaluation. These sequences, together with their ground truth, are made freely available for the research community.

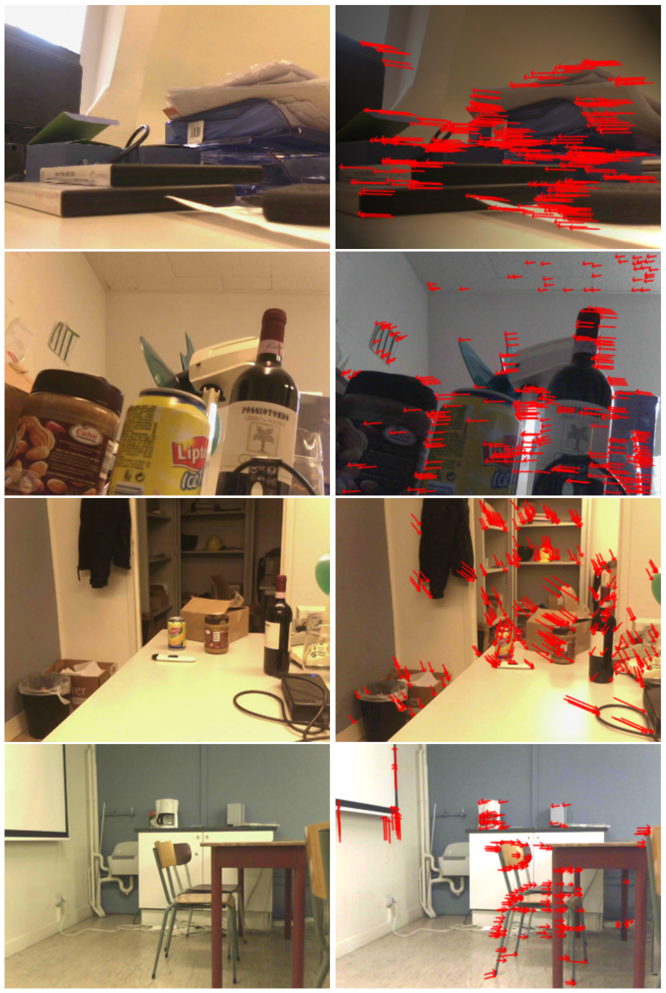
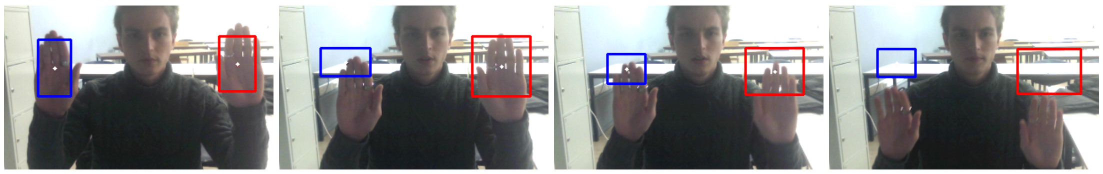


Fig. 6. Proposed video dataset with annotated ground truth flow

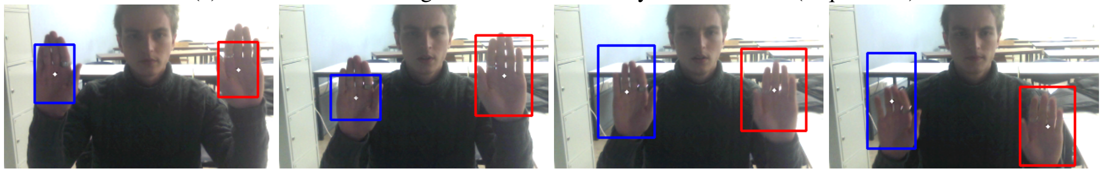
The proposed algorithm is used to improve the motion model of a particle filter based hand tracker. We incorporated the optical flow estimation in the state-of-the-art object tracker proposed by Spruyt *et al.* [14] and compared the tracking accuracy with the same tracker using a constant velocity motion model, which is a widely used approach in object tracking. The algorithms were evaluated using the publicly available dataset that was also used by Spruyt *et al.*

Figure 7 shows the widely used VOC error measure for each of the 8 video sequences used for evaluation. These results show that the error is lower for each of the sequences if optical flow is used to steer the motion model. Especially for sequence 4, which contains fast motion, camera movement and changing lighting, optical flow incorporation greatly improves the tracking performance.

Figure 8 shows several frames of sequence 4, used for evalua-



(a) Particle filter tracking with constant velocity motion model (sequence 4)



(b) Particle filter tracking with optical flow based motion model (sequence 4)

Fig. 8. Motion model evaluation for particle filter tracking

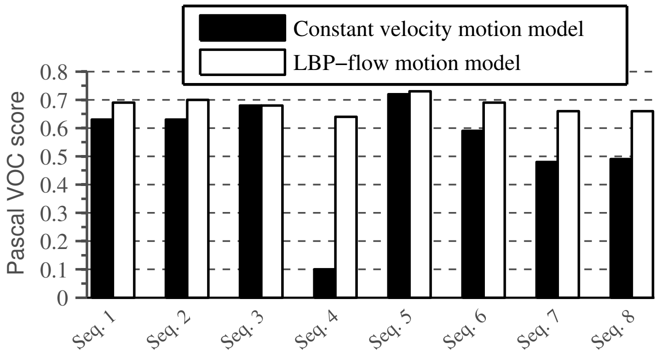


Fig. 7. Motion model evaluation for particle filter tracking

tion. The top row illustrates the results obtained using a constant velocity motion model, while the bottom rows shows the results obtained using our proposed optical flow solution.

7. CONCLUSION

A fast algorithm was proposed to calculate a sparse optical flow field. Regularization is performed iteratively, based on median filtering and concepts as used in Voronoi interpolation. The resulting method is illumination invariant and able to handle large displacements making it a perfect candidate for incorporation into real-time applications for uncontrolled environments such as object tracking.

The method was evaluated thoroughly, and appears to be much more robust than a pyramidal Lucas Kanade implementation while yielding similar or better results then a SURF-descriptor based tracker. Furthermore, the algorithm outperforms the state-of-the-art in Variational optical flow calculation, on a real-life video dataset.

Finally, we showed that the incorporation of our optical flow method into a state-of-the-art tracker greatly improves its robustness.

Our optical flow dataset is made publicly available together with its carefully annotated ground truth.

8. REFERENCES

[1] L. Xu, J. Jia, and Y. Matsushita, “Motion detail preserving optical flow estimation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 9, pp. 1744 –1757, sept. 2012.

[2] S. Baker, S. Roth, D. Scharstein, M. J. Black, J. P. Lewis, and R. Szeliski, “A Database and Evaluation Methodology for Optical Flow,” 2007, pp. 1–8.

[3] T. Kodama, T. Yamaguchi, and H. Harada, “A method of object tracking based on particle filter and optical flow to avoid degeneration problem,” in *SICE Annual Conference 2010, Proceedings of*, 2010, pp. 1529 –1533.

[4] M. J. Lucena, J. M. Fuertes, N. Perez de la Blanca, and A. Garrido, “Using optical flow as evidence for probabilistic tracking,” in *Proceedings of the 13th Scandinavian conference on Image analysis*, 2003, SCIA’03, pp. 1044–1059.

[5] M. Kristan, J. Perš, S. Kovačič, and A. Leonardis, “A local-motion-based probabilistic model for visual tracking,” *Pattern Recognition*, vol. 42, no. 9, pp. 2160–2168, 2009.

[6] M. Schikora, W. Koch, and D. Cremers, “Multi-object tracking via high accuracy optical flowand finite set statistics,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, may 2011, pp. 1409 –1412.

[7] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 32, pp. 105–119, 2010.

[8] T. Ojala, M. Pietikäinen, and T. Mäenpää, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 971–987, 2002.

[9] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid, “Description of interest regions with local binary patterns,” *Pattern Recognition*, vol. 42, pp. 425–436, March 2009.

[10] O. Grygorash, Y. Zhou, and Z. Jorgensen, “Minimum spanning tree based clustering algorithms,” *Tools with Artificial Intelligence, IEEE International Conference on*, vol. 0, pp. 73–81, 2006.

[11] D. A. Sinclair, “S-hull: a fast radial sweep-hull routine for delaunay triangulation,” July 2010.

[12] L. Alparone, M. Barni, F. Bartolini, and R. Caldelli, “Regularization of optic flow estimates by means of weighted vector median filtering,” *Image Processing, IEEE Transactions on*, vol. 8, no. 10, pp. 1462 –1467, Oct. 1999.

[13] V. V. Belikov, V. D. Ivanov, V. K. Kontorovich, S. A. Korytnik, and A. Y. Semenov, “The non-sibsonian interpolation : A new method of interpolation of the values of a function on an arbitrary set of points,” *Computational mathematics and mathematical physics*, vol. 37, pp. 9–15, 1997.

[14] V. Spruyt, A. Ledda, and W. Philips, “Real-Time Hand Tracking by Invariant Hough Forest Detection,” in *Proceedings of the IEEE International Conference on Image Processing ICIP’12.*, 2012, pp. 149–152.