

Towards robust automatic detection of vulnerable road users: monocular pedestrian tracking from a moving vehicle

Kristof Van Beeck¹, Floris De Smedt¹, Sander Beckers¹,
Lars Struyf¹, Joost Vennekens¹, Gorik De Samblanx¹,
Toon Goedemé^{1,2}, Tinne Tuytelaars²

¹ EAVISE, Lessius Mechelen - Campus De Nayer, Belgium

² ESAT - PSI - VISICS, K.U. Leuven, Belgium

kristof.van.beeck@mechelen.lessius.eu

Abstract

In this paper we present steps towards the automatic detection of vulnerable road users in video. Such a system can e.g. be used as an automatic blind spot camera for trucks. The aim of the system is to automatically warn the driver when the algorithm detects vulnerable road users in the camera images. Such an application implies severe time and performance constraints on the vision system, which introduce several challenges that need to be tackled. In this paper we described a first step towards such an intelligent detection system. We compare two different pedestrian tracking algorithms that we developed and implemented, with respect to the detection results and the applicability towards real-time performance. Both algorithms start from the same appearance-based pedestrian detector, and are based on a tracking-by-detection approach. The first algorithm uses the appearance-based detector in a Kalman filter motion model framework. The second algorithm uses the probabilistic detection result in a particle filter framework, to enable the use of multiple tracking hypothesis. We recorded two datasets: one with a standard camera from a moving vehicle and one recorded with a real blind-spot camera. At the moment evaluation of the algorithms is performed on the first dataset. To achieve real-time performance we implemented part of the appearance-based pedestrian detector onto dedicated GPU hardware using OpenCL. The speedup results that we achieved will also be discussed.

1 Introduction

This paper addresses the problem of the blind-spot zone involved with trucks. According to a report of the European Commission [2006], 1300 casualties are

Figure 1: Vulnerable road users are a very diverse class.



caused every year because of this blind spot zone. Research shows that the introduction of the blind spot mirror, obliged by law in the EU since 2003, did not decrease the number of casualties. Therefore we aim to develop an active vision system which warns the driver based on camera images from a blind spot camera. Existing active systems on the market today (e.g. ultrasonic sensors) cannot detect the difference between vulnerable road users and static objects. They generate false alarms and become annoying to the truck driver. We believe that a vision system can tackle these problems. Vulnerable road users however are a very diverse object class. Besides pedestrians, we also need to detect e.g. bicyclists, children, wheelchair users and mopeds. So our detection system has to deal with multi-class, multi-view and multi-pose object tracking (see figure 1). Since the field of view of the camera covers the blind spot area on the side of the truck, a camera image with a highly dynamic background is generated. Therefore standard computer vision techniques, such as background subtraction or background modeling, cannot be used here. Other challenges include the real-time character of the application. Only limited time is available to detect the vulnerable road users. This contradicts with the need for a high precision and recall rate. In this paper we present work towards such a fully automatic vulnerable road user detection system. We propose two pedestrian tracking algorithms, and compare them based on their accuracy, and on their speed of execution. Both algorithms start from an appearance-based detector which is based on the approach by Felzenszwalb et al. [2008]. They proposed a multi-part HOG model. Our main motivation for this choice is based on the work of Enzweiler and Gavrila [2009], who stated that this HOG model outperforms other state-of-the-art detection systems such as the wavelet-based AdaBoost cascade [Viola and Jones, 2001], combined shape-texture models and neural networks.

The first pedestrian tracking algorithm is based on a tracking-by-detection Kalman framework. Pedestrians are detected in a single frame, and using our motion model the next position is estimated. Based on this estimated next position in the consecutive frame, another appearance-based detection is done. If a match is found our tracker is updated. The second pedestrian algorithm uses a multi-hypothesis approach: the probabilistic outcome of the detector is integrated into a particle framework. In consecutive frames, a score is calculated for each particle. Based on this score the particles are reweighted. To achieve real-time performance, we implemented part of the Felzenszwalb HOG detector on a GPU architecture.

The paper is structured as follows: in section 2 related work concerning this topic is given. Section 3 give an in-depth overview of the two previously men-

tioned algorithms that we developed. Section 4 describes our experimental setup and the comparison results of our two algorithms. In section 5 we describe the results of our HOG detector implementation in OpenCL on dedicated GPU hardware. In section 6 we conclude this paper by giving some final remarks and a look on future work on this research topic.

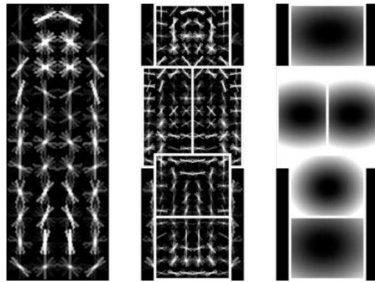
2 Related work

An extensive amount of research on object detection is available. Two well-know approaches are the Viola and Jones [2001] object detection framework, and the HOG model approach as introduced by Felzenszwalb et al. [2008]. Viola and Jones use the response of Haar-wavelets as weak classifier inputs for AdaBoost, which combines them into one or more strong classifiers. This results into an extremely rapid object detection framework. Lienhart and Maydt [2002] extended this idea by introducing tilted Haar-wavelet features. Messom and Barczak [2006] further generalized this concept into generic rotated Haar-wavelet features. Felzenszwalb et al introduced a new approach towards object detection. They extended the idea of Dalal and Triggs [2005], which use histograms of the orientation of the gradient (HOG) as object model, with a part-based model. According to Enzweiler and Gavrilu [2009], the Wavelet-AdaBoost combination is fast to calculate with medium accuracy, while the HOG model offers excellent accuracy results at the cost of the high calculation complexity. Therefore we used the HOG approach as our initial detector, and tackle the computational complexity by an implementation into dedicated GPU architecture. Concerning pedestrian tracking frameworks, most of them are based on a fixed background [Viola et al., 2005, Seitner and Hanbury, 2006], use a multi-camera setup [Gavrila and Munder, 2007] or cannot be used in multi-pedestrian situations. We want to develop a real-time monocular system which can detect pedestrians from a moving camera. To achieve this, we use two model estimation techniques: a Kalman filter [Kalman, 1960] and a particle filter [Arulampalam et al., 2002]. Existing tracking frameworks based on particle filters often use simple target models which are fast to calculate, for example color distributions [Nummiaro et al., 2002]. These techniques are only suitable in situations with minimal occlusion, and when the appearance of the object only slightly changes. We introduce the use of a very robust appearance-based pedestrian detection algorithm as an input for a particle filter framework. This results in reliable tracking even in harsh conditions.

3 Algorithms

In order to achieve high recall and precision rates, both algorithms that we developed rely on the same initial appearance-based pedestrian detection algorithm. Therefore we will briefly explain this approach, followed by the two algorithms that we developed.

Figure 2: The HOG model of a pedestrian. Left: root filter, middle: part filters, right: prior estimation of the position of the parts.

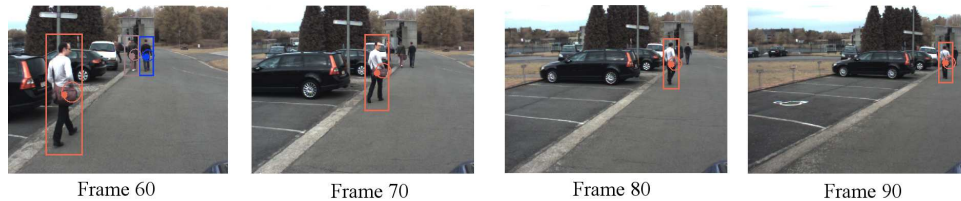


3.1 Initial HOG pedestrian detector

As initial pedestrian detector we use the object detection algorithm from Felzenszwalb et al. [2010b,c]. Their model consists of a root filter, part filters and a prior estimation of the position of the different parts with respect to the root filter. This generic approach can be used to detect a wide variety of objects, given enough input images to train the HOG model. Figure 2 shows the HOG model of a pedestrian. After the model is constructed, object detection is performed by building a scale-space pyramid of the input image to incorporate scale invariance. For each scale the HOG feature is calculated. One then computes the response of the root filter together with the transformed response of the different parts, which is calculated at twice the spatial resolution of the root filter itself. This approach leads to a very robust object detector. The main disadvantage of this detector is its computational complexity. Since one needs to look for objects at each scale and position for both the root filter and the different parts, detection is a time-consuming step. Recently the authors proposed a new detection object cascade [Felzenszwalb et al., 2010a]. The main idea is that one first looks for a weaker model and if this passes the model is enriched with more information, thus enabling fast rejection. This is somewhat similar to the Haar-wavelet/AdaBoost object detection cascade.

3.2 Kalman Tracking-by-detection framework

The first algorithm we developed is a tracking-by-detection framework using a Kalman motion model. It works as follows. An initial search region is defined in an image, where one expects that pedestrians enter the frame. In this search region we run the pedestrian cascaded HOG model. If a pedestrian is found, the centroid of this detection is calculated, and a Kalman filter motion model is instantiated. Using the motion model, the estimated position of the next centroid is calculated. Around this estimated position a circular region, of which the radius depends on the scale of the detection, is defined. Detections closer to the horizon are given a smaller radius. Finally the new search region in which we need to look for a detection in the consecutive frame is calculated. In the case of multiple pedestrian detections overlapping search regions are combined to reduce computation time. To further increase robustness we introduced some

Figure 3: Example tracking sequence using Kalman filter

application specific constraints. Detections above the horizon and detections of which the ratio of the bounding box is inconsistent with the expected scale in the image are discarded. To track the pedestrians in subsequent frames, we run the HOG detector on the new search regions. If a centroid of a pedestrian is found in the previously calculated circular region they are matched, and the Kalman motion model is updated with this information. In the case of multiple matches we match the one that yields the smallest Euclidean distance. If no match is found, the tracker is updated using the estimated position of the Kalman filter. If for multiple frames no match is found, the tracker is discarded. For each new detection, a motion model is instantiated. Figure 3 shows an example of our pedestrian tracker. More details can be found in Van Beeck et al. [2011].

Our Kalman model is based on a constant velocity motion model. Our experiments yield that this is a valid choice. Our initial Kalman motion vector is the opposite of the motion vector of the camera, since we can assume that the walking speed is negligible compared to the speed of the truck itself. As our state vector the position and velocity vector is used:

$$x_k = [x \quad y \quad v_x \quad v_y]^T \quad (1)$$

Only the position is observable. The Kalman filter is implemented with the following time update equation: $\hat{x}_k^- = A\hat{x}_{k-1}$. Here \hat{x}_k^- refers to the *a priori* state estimate at timestep k , while \hat{x}_k refers to the *a posteriori* state estimate at timestep k . Since we use a constant motion model our process matrix A becomes:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

The use of this Kalman filter framework allows for a near real-time implementation at very high accuracy, as we will show in section 4.

3.3 Probabilistic Particle-based framework

Using a Kalman filter we can only keep track of one specific hypothesis. This can be a disadvantage, especially in case of challenging environments. Since the

Figure 4: The likelihood of there being a person at that position in the image



HOG detector calculates a score which defines how high the detection matches the pedestrian model, we can use this score into a probabilistic framework. This allows multi-hypothesis tracking, thereby increasing the accuracy. We used a particle filter [Arulampalam et al., 2002] to implement this probabilistic framework. Particle filters approximate the probability density function $p(x_{0:t}|z_{1:t})$ of the state vector using a weighted set of discrete samples:

$$p(x_{0:t}|z_{1:t}) = \sum_{i=1}^{N_s} w_t^{(i)} \delta(x_{0:t} - x_{0:t}^{(i)}) \quad (3)$$

With normalized weights ($\sum_i w_t^i = 1$). The samples are drawn from an *importance sampling distribution* $\pi(x_{0:t}|z_{1:t})$, and the importance weight are calculated using:

$$w(x_{0:t}) = \frac{p(x_{0:t}|z_{1:t})}{\pi(x_{0:t}|z_{1:t})} \quad (4)$$

In practice one uses the prior as importance density $\pi(x_t|x_{t-1}^{(i)}, z_t) = p(x_t|x_{t-1}^{(i)})$ such that the weights are calculated using:

$$w_t^{(i)} = w_{t-1}^{(i)} p(z_t|x_t^{(i)}) \quad (5)$$

Each sample, called a particle, is propagated using a specific motion model. When a new detection is found, the discrete sample set is updated based on the observation and resampled. Each particle thus correlates with a specific hypothetical state. The weight of each particle indicates how important the particle is. As motion model we used the same model as used for the Kalman filter implementation. To reweight the sample set, we calculate for each particle the likelihood of there being a person at that location $p(z_t|x_t^{(i)})$, using the HOG model outcome. Figure 4 shows such a calculated likelihood heat map. Figure 5 gives a tracking sequence using the particle filter approach. In the next section we will discuss accuracy and speed results with this framework.

Figure 5: Example tracking sequence using Particle filter

4 Experiments & Results

To evaluate our algorithms, we recorded two datasets. The first dataset is recorded from a moving vehicle with a standard camera, and consists of about 2000 frames in which both pedestrians and bicyclists are visible. The second dataset is recorded with a real blind-spot camera from a static truck and only pedestrians are recorded, for a total of 10000 frames. Both datasets are divided into short sequences: the first dataset contains 12 sequences while the second dataset contains 7 sequences. The frame rates of both cameras equals 15 FPS at a resolution of 640×480 . Figure 6 shows the blind-spot camera that we used for our second dataset. It has a viewing angle of 115 degrees, thus the resulting images have a high degree of distortion at the borders. Figure 7 gives one frame of each of the two datasets to get an idea of the camera position and distortion. Note the large distortion in the right image. At the moment evaluation of the algorithms is done using the first dataset.

Our algorithms are implemented in Matlab and partially in C. In this evaluation results no OpenCL optimizations are used for the moment. For the particle filter we used 10 particles in our evaluation. The results are computed on an Intel Xeon Quad Core with a clock speed of 3 GHz, using an unparallelized single-threaded manner. When we run the standard Felzenszwalb pedestrian detection software as given by Felzenszwalb et al. [2010c] over an entire frame detection time equals on average 1.17 seconds (0.86 FPS). Table 2 show the speed results of our tracking algorithms on the first data set, averaged over all sequences in that dataset. Table 1 shows accuracy results for the Kalman filter implementation (Van Beeck et al. [2011]). We notice that our Kalman tracking algorithm greatly reduces the computation time because of the reduced search space, while maintaining both high precision and recall rates. The particle filter implementation has the advantage that one can model both non-linear systems and non-gaussian distributions. A disadvantage is that for each particle one needs to recalculate its

Table 1: Accuracy results for Kalman Filter approach.

	avg. fps	precision	recall
seq. 1	9.57	0.76	0.92
seq. 2	8.97	0.95	0.93
seq. 3	9.47	0.95	0.8
average	9.34	0.89	0.88

Table 2: Speed Results for both our algorithm implementations

		Dataset 1
Kalman Filter Approach	Average max. FPS	12.6 FPS
	Average min. FPS	2.8 FPS
	Average FPS	8.6 FPS
Particle Filter Approach	Average max. FPS	0.25 FPS
	Average min. FPS	0.11 FPS
	Average FPS	0.22 FPS

Figure 6: Our blind spot camera mounted on the test truck.

weight. To maintain a reasonable complexity only simple weight recalculations can be used, such as color histograms. In our implementation we used the HOG model to recalculate the weights, to increase invariance to object variations. This however comes at the cost of high computational complexity, which lead to low frame rates, as can be seen in table 2. In the future we hope to decrease the computational time using the OpenCL implementation, and use the particle filter framework as our default detector.

5 Implementation

To achieve real-time execution times, we are developing an OpenCL implementation for the Felzenszwalb star cascade object detection algorithm. In this section we present the results of our ongoing work. The code we use is based on their implementation, no optimizations are performed which affect the precision of the results. To generate reliable statistics, we averaged the timing over 100 runs. The GPU implementation is executed on a dedicated NVIDIA GTX295 card. In table 3 the specifications of this card are given. The CPU execution is performed on a Intel core i7 870 processor with 2.93GHz.

In section 3 we gave a brief overview of how the object detection is done. First a feature pyramid is built, and one search for the object model on each layer of the pyramid. To construct this feature pyramid the source image is rescaled multiple times, and a scale-space pyramid is constructed. For each image in the resulting image pyramid, the HOG-features are calculated. Section 5.1 details

Figure 7: Example frames from our datasets. Left: standard camera, Right: real blind-spot camera



the image rescaling step, while section 5.2 goes more deeply on the feature computation step.

5.1 Image Rescaling

First, the image must be rescaled to build a scale-space pyramid. The scaling factor is $2^{1/n}$ where n determines the number of layers in the pyramid. Each image is rescaled from the original image, thus the rescaling process is independent of other images, which makes it parallelizable. Since the source image is used multiple times, we put this in texture memory so we can profit from caching. The source image is kept in integer format to minimize the data transfer to the GPU. Because it is smaller, the number of pixels we can cache increases. Transferring the image to the GPU takes approximately $11\mu s$. The rescaling of the image is done in two steps. First we rescale in horizontal direction, afterwards in vertical direction. In both steps the rescaling for each row (or column in the case of vertical rescaling) is based on the same pixels. Thus, for each scale we calculate two arrays which define which pixels to use in a row/column and use these to rescale all rows/columns in parallel. Figure 8 shows the timing performance of our GPU resize implementation. In the first scale no rescaling is performed, hence this is very fast. Because the size of the arrays decreases with the image scale the computational time also decreases, since less calculations need to be performed. This is clearly visible for scales 2 to 10.

Since each pixel of the resulting image can be calculated independently, we could calculate all of these pixels in parallel. This is done by launching one thread per destination pixel. In figure 10 the resulting calculation time is given. Because of the high degree of parallelism a large improvement is noticed over the previous approach, and no CPU work has to be done in advance. However,

Table 3

OpenCL version	1.0
CUDA cores	480 (240 per GPU)
compute units	30
Clock frequency	1242MHz
Global memory	1792MB (896MB per GPU)

Figure 8: GPU time needed to calculate pixels of a resized image on different scales, using one thread per row/column.

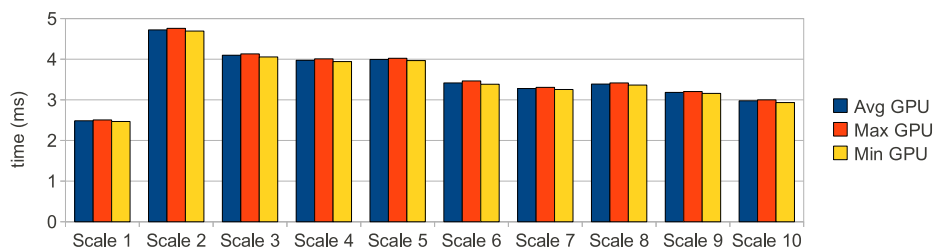
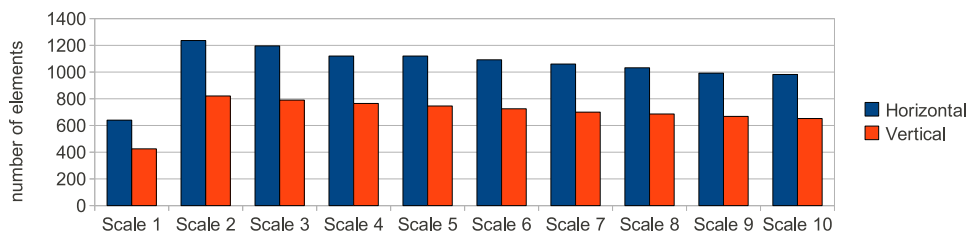


Figure 9: Number of elements in the arrays used to rescale the images.



one can note that the improvement is not proportional to the increased number of threads. There is still room for further improvement.

The goal of the implementation on GPU was to improve the time needed by the CPU to construct the image pyramid. Figure 11 shows the CPU calculation time, compared with both GPU implementation times. Notice the huge difference in time performance: while the first GPU implementation requires only about $3.5ms$ per scale, the CPU consumes $30ms$ per scale. To further reduce the calculation time using the one pixel per thread implementation the GPU uses only about $400\mu s$. This illustrates the big time performance increase one can get from a GPU implementation. The shape of the curve is similar as for the GPU. One can see in figure 9 that going from scale 1 to scale 2 the size of the array doubles. Therefore on GPU the calculation time is doubled, on CPU this is not the case. This is due to the type of memory. On GPU the amount of accesses to global memory forms a bottleneck for speed where on CPU this influence is far less noticeable since more time is spent on calculation than on memory transfers.

5.2 Feature calculation

To construct the feature pyramid, we have to calculate the HOG features for each image of the scale-space image pyramid. The calculation of the HOG features consist of two parts: first a histogram of gradients is created, and then these gradients are used to calculate the feature values themselves. We also implemented this second part on GPU. The resulting feature pyramid consists of 32 layers, and each thread calculates one value in each layer. The comparative CPU versus GPU timing results are given in figure 12. We can notice the decreasing calculation time with the size of the image we have to process. Here we also notice a huge

Figure 10: GPU time needed to calculate pixels of a resized image on different scales, using one thread per pixel.

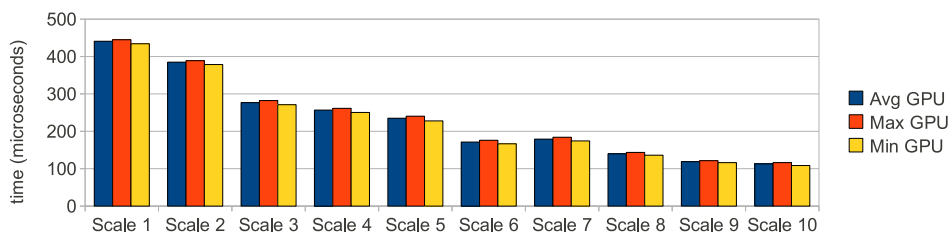
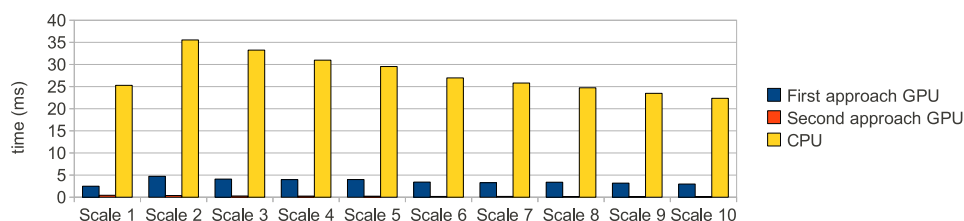


Figure 11: CPU time versus GPU time needed to calculate pixels of a resized image for different scales.



improvement of the GPU implementation over the CPU implementation. These results are calculated without taking into account the time needed to transfer the histograms to GPU. The reason for this is that in a later implementation we will implement the whole feature pyramid calculation on GPU so only the final pyramid should be transferred back to host memory.

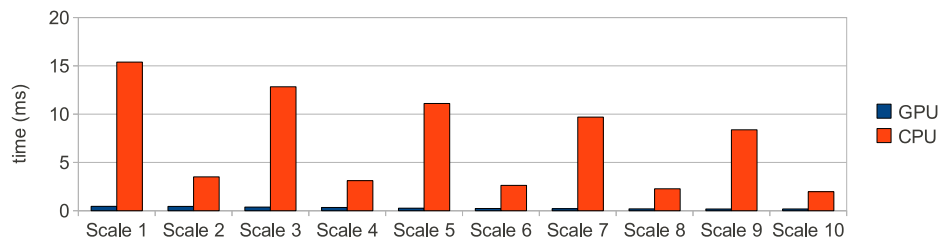
5.3 Conclusion on GPU implementation

Table 4 summarizes the timing results of our implementation of parts of the Felzenszwalbs [Felzenszwalb et al., 2010a] algorithm on CPU and GPU. As can be seen, it is a work in progress, but the most computationally intensive parts show a huge speedup when implemented on GPU. Since the layers of the scale-space pyramid can be calculated independent of each other, we can run the GPU parts in parallel. To make a fair comparison with the CPU implementation, we also keep the GPU implementation sequential.

Table 4: Summary of timing results for sequential calculation of 10 scales

	CPU (ms)	GPU (ms)
Rescale	277.93	35.519
		2.316
Features	70.89	2.86

Figure 12: CPU time versus GPU time needed to calculate the features out of the histograms.



6 Conclusion and Future Work

We presented the unique combination of a very robust appearance-based detector and two tracking frameworks: one based on a Kalman filter approach and one based on a Particle filter. Experiments on self-recorded datasets show promising results towards accuracy and speed measurements for the Kalman implementation. Furthermore we presented our ongoing work on the implementation of this appearance-based detector into dedicated GPU hardware, to further speedup our tracking algorithms. Future work involves the implementation of the remaining parts of the algorithm in GPU, and the integration of this implementation into our tracking frameworks.

Acknowledgements

This work is partially supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), Belgium.

References

- M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.
- N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, June 2005.
- M. Enzweiler and D. Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2179–2195, 2009.
- P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

- P. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010a.
- P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9), 2010b.
- P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. <http://people.cs.uchicago.edu/~pff/latent-release4/>, 2010c.
- D. Gavrila and S. Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International Journal of Computer Vision*, 73(1):41–59, 2007.
- R. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME Journal of Basic Engineering*, 82:35–45, 1960.
- R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, pages 900–903, 2002.
- C. Messom and A. Barczak. Fast and efficient rotated haar-like features using rotated integral images, 2006.
- K. Nummiaro, E. Koller-meier, and L. Van Gool. A color-based particle filter. 2002.
- Report of the European Commission. Commission of the european communities, european road safety action programme: mid-term review, 2006.
- F. Seitner and A. Hanbury. Fast pedestrian tracking based on spatial features and colour. In *Proceedings of the 11th Computer Vision Winter Workshop*, pages 105–110, 2006.
- K. Van Beeck, T. Goedemé, and T. Tuytelaars. Towards an automatic blind spot camera: Robust real-time pedestrian tracking from a moving camera. In *Proceedings of the 12th IAPR Conference on Machine Vision Applications (to appear)*, 2011.
- P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.
- P. Viola, J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.