Pedestrian Detection at Warp Speed: Exceeding 500 Detections per Second

Floris De Smedt*, Kristof Van Beeck*, Tinne Tuytelaars and Toon Goedemé
EAVISE, ESAT-PSI-VISICS, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract

Object detection, and in particular pedestrian detection, is a challenging task, due to the wide variety of appearances. The application domain is extremely broad, ranging from e.g. *surveillance to automotive safety systems. Many practical applications however often rely on stringent real-time processing speeds combined with high accuracy needs. These demands are contradictory, and usually a compromise needs to be made. In this paper we present a pedestrian detection framework which is extremely fast (500 detections per second) while still maintaining excellent accuracy results. We achieve these results by combining our fast pedestrian detection algorithm (implemented as a hybrid CPU and GPU combination) with the exploitation of scene constraints (using a warping window approach and temporal information), which yields state-of-the-art detection accuracy. We present profound evaluation results of our algorithm concerning both speed and accuracy on the challenging Caltech dataset. Furthermore we present evaluation results on a very specific application showing the full potential of this warping window approach: detection of pedestrians in a truck's blind spot zone.*

1. Introduction

A pedestrian detector which is at the same time fast and accurate would open up a wide variety of applications, including robotics, surveillance and automotive safety. These applications clearly benefit from the high robustness most recent algorithms can achieve. Indeed, over the past few years impressive accuracy improvements were obtained on challenging benchmark datasets, containing a wide variety of poses and appearances. Unfortunately, high accuracy often comes at the cost of high computation time, making these algorithms unfeasible in real-life applications. We overcome this problem via algorithmic optimization and via the exploitation of scene constraints. We present an efficient pipelined hybrid CPU/GPU pedestrian detector implementation. While being fast on its own, we further improve the speed-up using both the *warping window approach* we proposed in [18], and the integration of temporal (tracker-based) information. This warping window approach allows to limit the search space and thereby reduces both the false positive rate and the detection time while allowing arbitrary non-linear camera distortion and extreme viewing angles. We benchmark our hybrid pedestrian detector implementation both with respect to accuracy as to speed on the Caltech dataset [9, 10], and show that despite the speed-up, we achieve state-of-the-art accuracy. Moreover, we quantitatively demonstrate how the warping window approach further increases the accuracy on the Caltech dataset. Finally, we propose experiments and results on a second dataset targeting a demanding application: the detection of pedestrians in the blind spot camera images of a truck. Using this application we illustrate the full potential of this warping window approach, and achieve excellent accuracy results with very high processing speeds (500 detections per second). Such speeds are useful for *e.g.* crowded scenes or when implementing on embedded hardware with limited processing power. This paper is structured as follows. We first describe related work in section 2. In section 3 we describe our hybrid CPU/GPU pedestrian detector, while in section 4 we discuss the warping window approach with quantitative accuracy improvement results. We then combine both approaches in section 5, and show how our speed-up is realized. We conclude this work in section 6.

2. Related work

Excellent and detailed evaluation benchmarks of recent pedestrian detectors can be found in the literature [9, 10, 11]. Here we briefly discuss only the most relevant ones. Initially Dalal and Triggs [5] proposed the *Histograms of Oriented Gradients* (HOG) pedestrian model for human detection. To further improve the accuracy their idea was extended by Felzenszwalb *et al.* to a part-based HOG model [14]. Increasing the complexity of the model evidently leads to higher computational complexity and hence lower processing speeds. To overcome this burden the same authors presented a cascaded version of their object detec-

* F. De Smedt and K. Van Beeck contributed equally to this paper.

tion scheme [13], in which pruning of false detections is achieved early in the detection stage (much like proposed by [19] in their work using Haar wavelets). These algorithms employ a so-called *sliding window approach*: one searches across all possible scales and locations in the image. This is challenging when targeting real-time applications. To cope with this, several algorithmic optimisations have been proposed. Lampert *et al*. [16] present a branch and bound scheme to reduce calculation time. Dollár *et al*. propose their *Fastest Pedestrian Detector in the West* (FPDW) [7] where intermediate feature responses are approximated from feature responses of nearby scales, eliminating the need to construct a full feature pyramid. Based on this approach, in [2], Benenson *et al*. propose work in which they perform model rescaling instead of image rescaling to speed-up detection. Their detector is based on [8] and combined with their stixel world approximation [1], they achieve fast pedestrian detection. Recently the authors proposed their *Roerei* detector which achieves state-of-the-art accuracy results [3]. We aim to develop a pedestrian detector solely on monocular information, and exploit scene constraints combined with a fast hybrid CPU/GPU implementation to improve both accuracy and calculation time. In [17] Prisacariu and Reid propose a fast GPU implementation of the standard HOG model, coined *fastHOG*, which is closely related to our work. Since our work is based on the part-based pedestrian detector from Felzenszwalb *et al*. [12], we achieve much higher accuracy results. Even when using no scene constraints, our pedestrian detector implementation is slightly faster. Concerning the use of scene constraints, Cho *et al*. achieve real-time pedestrian detection results on the Caltech dataset using standard ground-plane assumptions [4]. However, our warping window approach as proposed in [18] is more flexible as compared to this (as illustrated in section 4), and our fully integrated implementation outperforms theirs with respect to processing speeds.

3. Hybrid Pedestrian Detector

Speed improvement can be obtained in two ways: either optimize the execution or decrease the search space (or a combination of both). In this section we discuss the first, while in the next section (section 4) we target the latter. The accuracy and speed results discussed in this section therefore are measured without scene constraints, thus applying a full scale-space search. The object detection algorithm we start from is based on the *cascade latent SVM detector* proposed by Felzenszwalb *et al*. [14]. This detector achieves state-of-the-art accuracy results. Their algorithm works as follows. To be able to detect pedestrians at multiple scales, a first step is the construction of a scale-space pyramid (which is model independent). For each layer, feature responses are computed resulting in a feature pyramid. These feature responses are based on the HOG features from [5]. A second step consists of the actual pedestrian detection in this feature pyramid (a model evaluation step). This is done by matching a pretrained HOG model (visualized in figure 1) in each layer of the feature pyramid. The model consists of a HOG root filter (fig. 1 left), representing the rough shape of a pedestrian, and several parts representing the head and limbs (fig. 1 middle). The position of these parts are latent variables. A deformation cost (fig. 1 right) penalizes each part based on its deviation from the optimal location, allowing for a spring-like interaction with the root filter. The score of the root model at a specific scale is combined with the score for each part at twice the resolution, resulting in a final detection score. Using this original part-based pedestrian detector implementation [12] as a baseline, we propose two new implementations. In the first implementation we ported the calculation of the feature pyramid to the GPU, resulting in a significant speed-up. Our second implementation consists of a multi-threaded hybrid CPU/GPU implementation, achieving a speed-up of $12.7\times$ over the original implementation. In section 5 we then integrate this last implementation with scene constraints and temporal information, and propose our *WSPD* (*Warp Speed Pedestrian Detector*), which achieves pedestrian detection at 500 detections per second. For the remainder of this section we discuss our first implementation (feature pyramid on GPU) in subsection 3.1, the multi-threaded hybrid CPU/GPU implementation in subsection 3.2, and give evaluation results concerning both speed and accuracy in subsection 3.3.

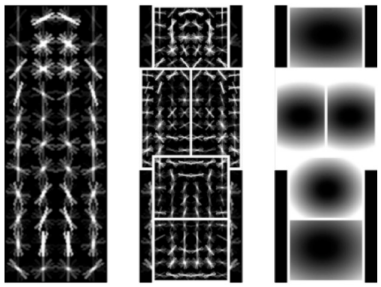


Figure 1. The proposed HOG model from [10]. Left: root filter. Middle: HOGs of parts. Right: deformation costs for each part.

3.1. GPU Feature pyramid implementation

The feature pyramid consists of multiple layers, each containing the features of a rescaled version of the source image. The publicly available implementation (referred further as *LatSVMV4 (original)* - referring to the latent variables) does not suffice for real-time applications. Therefore as a first step towards a GPU implementation we reimplemented the algorithm to C++. Since GPU hardware benefits from the use of floats, our C++ reimplementation (further referred to as *WSPD-v0.1 (float)*) uses *float* variables (as opposed to the original *double* implementation). One could argue that this leads to a decrease in accuracy. We show

however, that this is not the case (see subsection 3.3 below). Based on this C++ implementation we implemented the feature pyramid calculation on GPU using CUDA. In essence CUDA is a C extension that allows the use of Nvidia GPU hardware as an execution device, enabling faster execution of algorithms that use data parallelism. In previous work we implemented the feature pyramid in OpenCL [6], however the new implementation proposed here is faster due to the use of this hardware specific language. We further refer to this GPU implementation as *WSPD-v0.2 (GPU)*.

3.2. Multi-threaded hybrid implementation

Although the use of GPU hardware allows for a significant speed-up, it does not fully exploit the capabilities of the hardware system, since the CPU is only active when the GPU is idle and vice versa. To further speed-up the algorithm, and to circumvent this problem, we propose an implementation using a pipelined object detection scheme. In particular, we calculate both the feature pyramid of a frame (on GPU) while at the same time performing the model evaluation step for each layer in the feature pyramid from the previous frame (on CPU). This way the CPU and GPU are both active as a hybrid system, allowing an increased detection throughput. Besides running the feature pyramid and model evaluation in parallel on CPU and GPU, we can further increase the detection throughput by matching the execution time of each step in the pipeline. The feature pyramid *e.g.* is almost twice as fast as the model evaluation step, so for each feature pyramid process we run two model evaluation processes, leading to a higher speed. An even faster detection throughput is achieved when running multiple instances of the detection pipeline in parallel. Figure 2 shows a schematic overview of our implementation structure, which we further refer to as *WSPD-v0.3 (hybrid)*. This structure focusses on the use of eight threads (one per block). As shown, we use two detection pipelines in parallel. We will discuss each block of the schematic overview in more detail below.

Preprocessing Here, all preprocessing (*e.g.* cropping, rescaling, rotating and search space pruning) is performed. The results are placed in the image queue for further processing.

Feature Pyramid In this block the feature pyramid is calculated. The images are gathered from the image queue, while the feature pyramids are pushed into the pyramid queues. Since the calculation of the feature pyramids is executed on GPU, the data transfer to the GPU is also included here. In the schematic overview of fig. 2, two instances of the feature pyramid are displayed, indicating that the feature pyramid of two frames is computed in parallel. Due to the modular approach of our innovative

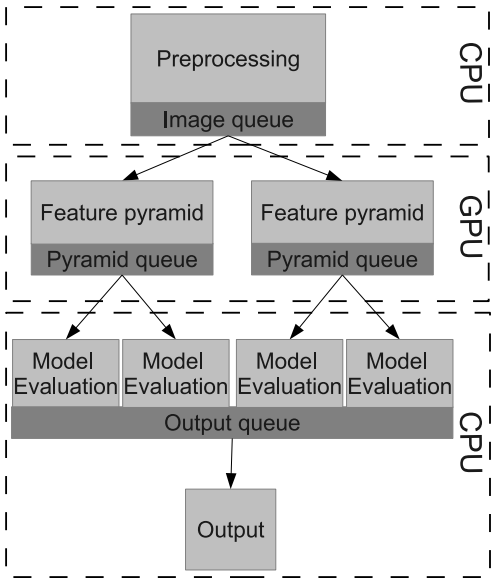


Figure 2. A schematic overview of the hybrid detector.

hybrid detecting scheme the implementation is easily hardware scalable. In our experiments each feature pyramid instance ran on a seperate CUDA device (our GPU includes two such devices). Although it is possible to run multiple instances on the same CUDA device, this is evidently limited by the GPU resources.

Model Evaluation For each calculated feature pyramid, we need to evaluate the pretrained model on each layer in the pyramid. Each feature pyramid feeds two model evaluation processes: this is needed since the feature pyramid calculation is almost twice as fast as the model evaluation. Since the feature pyramids are independent of the model to be detected, each instance of the model evaluation block is able to search for another model, thereby reducing calculating time if one aims to detect multiple object classes at once.

Output This final block gathers all the processed results and performs the post processing such as NMS (*non-maximum suppression*), reordering the frames, displaying detections and saving the frames.

3.3. Evaluation

We performed thorough experiments concerning both speed and accuracy of our implementations. Note that, at this point, all experiments are still performed without scene constraints, which we introduce in section 4. The speed comparison of our and publicly available implementations is given in table 1. All speed measurements were obtained on the same hardware (Intel Core i7 CPU 965 @ 3.20GHz with 12GB RAM and one Nvidia GTX295 GPU). Our experiments were performed on both the original image size (640×480) as well as on an upscaled version (1280×960).

	640 × 480	1280 × 960	Speed-up
LatSVMV4 (orig.)	1.33 FPS	0.33 FPS	1×
WSPD-v0.1 (float)	1.6 FPS	0.37 FPS	1.12×
WSPD-v0.2 (GPU)	2.97 FPS	0.85 FPS	2.58×
WSPD-v0.3 (hybrid)	12.9 FPS	4.17 FPS	12.7×
fastHOG	10.6 FPS	2.67 FPS	

Table 1. Speed results of our warp speed pedestrian detectors compared to the public implementation of the algorithm and fastHOG. We upscaled the image (×2), needed to detect pedestrians with a height around 50 pixels. No scene constraints are applied yet

This upscaling is needed if one wants to detect small pedestrians (around 50 pixels) since the HOG model is optimised for pedestrians with a height around 100 pixels. These upscaled image versions are used for our benchmark. We compared the relative speed-up we obtained to the original algorithm and fastHOG [17], a publicly available fast implementation of the HOG algorithm [5]. As can be seen in the table, our C++ reimplementation with *floats* is slightly faster than the original implementation. With our implementation of the feature pyramid on GPU we achieve a speed-up of $2.58\times$. While already faster, the overall speed-up is limited since we only implemented the feature pyramid on GPU. Our multi-threaded hybrid CPU/GPU implementation achieves a speed-up factor of $12.7\times$, thereby allowing real-time processing of 640×480 images (12.9 FPS). As mentioned in related work (section 2), the main motivation for using more advanced object detection algorithms in real-life applications is the increased accuracy. This however imposes a constraint on the allowed accuracy loss for fast implementations. Therefore we compared our reimplementations of the algorithm with existing algorithms on the publicly available Caltech dataset [9]. All experiments are performed on the challenging *reasonable* labeled pedestrians. Our results are visualized in figure 3, where we display the miss rate versus the false positives per image (FPPI). The

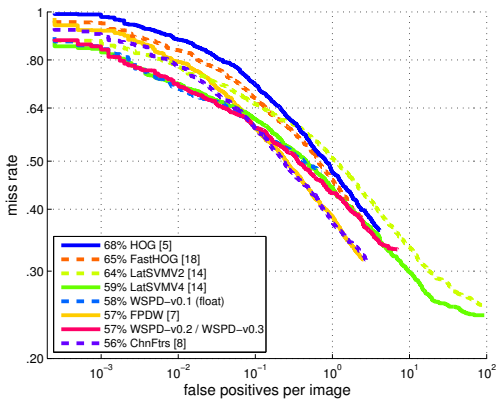


Figure 3. Comparison of existing detectors and our implementation. The results of HOG [5] and FPDW [7] are obtained from [10]

part-based detector (*LatSVMV4* [14]) achieves excellent accuracy results as compared to other state-of-the-art detectors (*e.g*. *FPDW* [7], *ChnFtrs* [8]), which was our motivation to use this detector as a baseline. These part-based detectors perform best on large-scale pedestrians [10]. However, detection accuracy loss on small pedestrians is minimal. Concerning our implementations, our speed-up does clearly not come at the cost of a performance drop.

4. Applying scene constraints (warping window approach)

As mentioned in section 3, besides a fast implementation, a speed improvement is achievable by decreasing the search space. Traditional object detectors use a sliding window paradigm, of which one of the bottlenecks is the large search space since at each position in the image every scale is evaluated. Often, ground plane assumptions are used for this matter [4]. They allow for a reduction in search space, while at the same time reducing the false positive rate. Such simple ground-plane assumptions have their limitations though: they cannot cope with *e.g*. non-linear camera distortions. Therefore, to speed-up detection we integrate our *warping window approach* that we introduced in [18] with our hybrid pedestrian detector and integrate temporal information to achieve higher processing speeds. This warping window approach can handle more complex scenarios, *e.g*. extreme camera viewpoints and wide-angle lens distortions. In this section we discuss the warping window approach, and quantitatively show how we use it to improve the accuracy on the Caltech dataset. In the next section (section 5) we then describe this integration and achieve extremely high detection speeds, even in the case of non-trivial camera viewpoints. In section 5, we illustrate this based on a challenging application: detecting pedestrians in the blind spot zone of a truck. In figure 4 an example input frame is shown from this dataset [18].

The key idea of the warping window approach is that, for each image position $\mathbf{x} = [x, y]$, we model the transformation caused by distortion locally. As can be seen in this example input frame, the pedestrians appear rotated and scaled. Assuming a flat ground-plane, their rotation θ and scale s only depends on the position in the image. Thus, if for each pixel position in the image the rotation and scale is known, detection time can drastically decrease. Instead of a full frame search, we can warp the region of interest and thereby eliminate the need for an entire scale-space search. Detecting pedestrians then consists of four steps: an extraction step (extract the ROI from the image), a warp step, running a detector on the warped image and finally retransform the detections to the initial image. This is visualized in figure 4. In the warp step the pedestrians are transformed to an upright position at a fixed height (140 pixels - as motivated in our previous work [18]), thus in the detection step

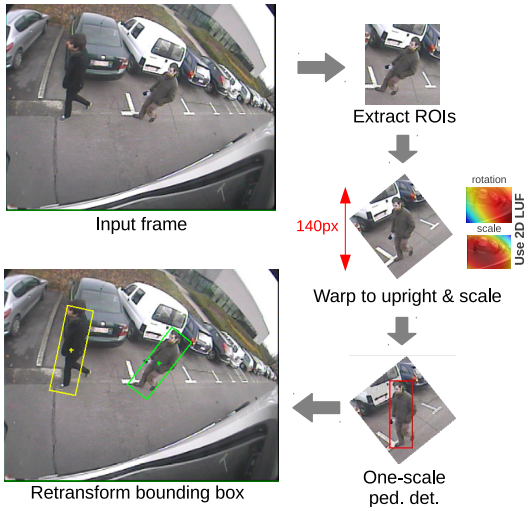


Figure 4. Illustration of the warping window approach [18].

a pedestrian detector is needed at only one specific scale, which obviously is very fast. If modelled with a rotation and scale, one can apply the warping window approach to pedestrians ROIs (I) using $I_{warp} = TI$ where transformation matrix T equals:

$$T = \begin{bmatrix} s\cos\theta & -s\sin\theta & 0 \\ s\sin\theta & s\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Note that this warping window approach is much more flexible as opposed to standard ground-plane assumptions. This approach is easily generalizable to complex arbitrary camera viewpoints, and situations where non-linear camera distortion and extreme viewing angles occur (*e.g.* surveillance cameras, wide-angle lenses and so on). A one-time calibration step is needed in order to obtain the scale and rotation value for each pixel position. These scale and rotation values are obtained by fitting 2D functions through manually labelled sparse sample points in the image during a calibration step (see figure 5), and thus form 2D look-up-functions (LUF). Let us now demonstrate how we can benefit from the warping window approach. Since no public datasets with large distortion are available, we

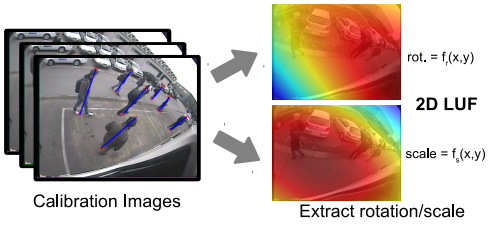


Figure 5. A one time calibration is needed, yielding the LUF for both the rotation and scale [18]

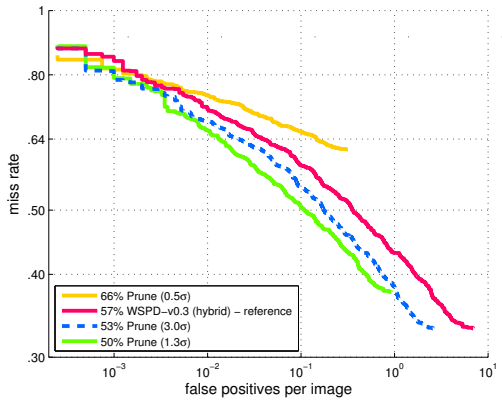
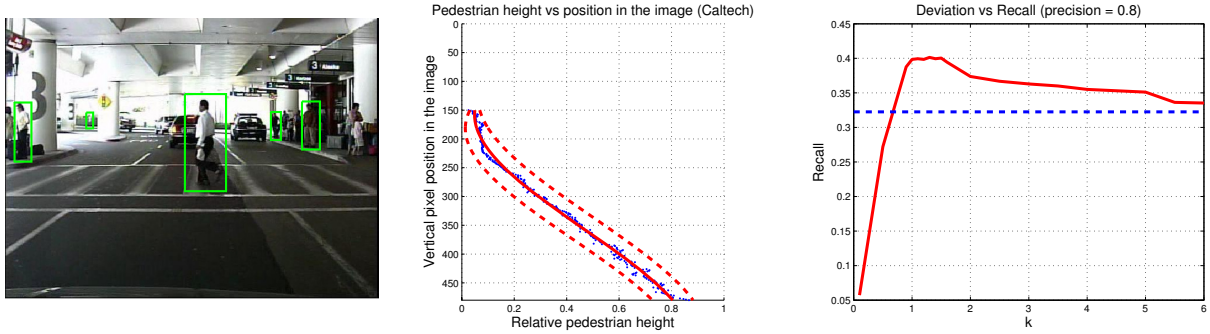


Figure 6. Accuracy improvement achieved on the Caltech dataset when using the warping window approach

first illustrate the gain in accuracy on the Caltech benchmark dataset. Note that here we use the warping window as a post-processing step to show the potential accuracy improvement, whereas the warping window approach normally is used as a pre-processing step to reduce the search space. Due to the basic camera viewpoint there, the full potential of the warping window is not yet exploited here. In section 5 we discuss the application of the warping window approach in a much more complex setting, and give quantitative speed results as well. As a reference, we start from the detection results of our multi-threaded hybrid pedestrian detector from section 3 (*WSPD-v0.3*) on the Caltech dataset. Here, due to the camera viewpoint no rotation is needed, hence we only model the pedestrian scale at each image position. Since the camera is positioned in a forward-looking manner, we assume that at each horizontal pixel line the scale is constant, reducing the dimensionality of the function to one in this case. Based on the labelled Caltech training data from [10] (see figure 7a for an example frame), we extract the height of each labelled pedestrian (normalized w.r.t. 480 pixels, the image height) at each pixel position, and average those observations per horizontal pixel line. Detections above the horizon are discarded. These datapoints are visualized in figure 7(b). Next we fit a third order polynomial function through these datapoints (solid red line). The dotted red lines illustrate two times the standard deviation (2σ) at each horizontal line. This transformation model can then be used to warp the ROIs to a fixed height. However, as noted above, here we use it to prune the results to show the potential accuracy improvement. If the height of a new detection is *inconsistent* from what is expected at that particular position, the detection is discarded. As a measure of inconsistency, we use the degree of deviation. If we allow much deviation no or only slight improvements are obtained since little pruning is applied, while on the other hand limiting the possible deviation too much leads to a significant drop in the recall rate. We empirically deter-



(a) Example Caltech frame with labeled pedestrians

(b) Pedestrian height vs y-position

(c) Influence of k-value on recall rate

Figure 7. The warping window approach applied to the Caltech dataset

mine the maximum allowed deviation (expressed as $k\sigma$) and evaluated the recall rate at a constant precision rate (80%). These results are displayed in figure 7(c). Here one clearly sees that the optimal deviation is found around 1.3σ, while at higher values of k the recall rate converges to the recall rate of the reference implementation (displayed as the dotted blue line). Figure 6 gives the miss rate versus the FPPI for both the original implementation along with three values of the deviation; an optimum is reached at 1.3σ. Applying the warping window approach thus leads to an accuracy improvement: at *e.g.* 0.1 FPPI, the miss rate decreases from 58% to 50%. In the next section we demonstrate how we use this approach to also increase the detection speed.

5. Warp Speed Pedestrian Detector

In this section we now present the combination of the previously discussed warping window approach (section 4) and the multi-threaded hybrid CPU/GPU implementation (section 3). As mentioned, this implementation achieves 12.9FPS without the use of scene constraints. Here we demonstrate the integration of the warping window approach and propose our *Warp Speed Pedestrian Detector* (WSPD) which achieves pedestrian detection at up to 500 detections per second, without loss in accuracy. We illustrate its potential on a challenging real-life problem: detecting pedestrians in the blind spot zone of a truck.

5.1. Single ROI selection

In section 4 we described how the warping window approach is used to reduce the search space. At each pixel location only one scale needs to be evaluated. To further reduce the search space, we can use techniques that limit the number of locations where a detection needs to be performed, based on some detection probability measure. In a fixed scene, simple and efficient background modelling techniques (*e.g.* background subtraction) can be used. However, in moving scenes with a highly dynamical background more complex techniques are necessary. One such technique is pedestrian tracking. If we are able to track the pedestrians throughout the scene, we can predict the search location (called tracking-by-detection), thereby reducing calculation time. Moreover we observe that in most applications the position where pedestrians enter the scene is predictable (*e.g.* doors). In the demonstrated blind spot application (discussed below), the Kalman tracker [15] has proven to be a robust technique for this purpose [18]. It predicts future positions based on detections in the past combined with a specific motion model. Based on this prediction, pedestrian ROIs are determined. For more detailed information on this we refer to [18].

5.2. Blind spot application

In most real-life computer vision applications, due to the specific position of the camera, distorted camera viewpoints occur. Due to its flexiblity, the warping window approach can cope with these specific distortion. To show its full potential we use an application with a challenging camera viewpoint: detection of pedestrians in blind-spot camera images. Because of the $115°$ wide-angle lens a specific distortion pattern occurs introducing non-linear camera distortion (see figure 4 for an example frame). Since the camera is mounted on a moving truck, we have to deal with a highly dynamic background. The LUFs (as described in section 4) are obtained by fitting 2D second order polynomial functions through sparse datapoints obtained during a one-time calibration step (fig. 5).

5.3. Pedestrian detection at Warp Speed

The crux of the matter is that we can use the warping window approach in combination with a Kalman tracker, and use the predicted search locations as input for our hybrid CPU/GPU pedestrian detector (section 3.2). This way we are able to perform pedestrian detection at unprecedented high speeds. Since we reduced the search space to a single scale and a single ROI, our detector only fo-

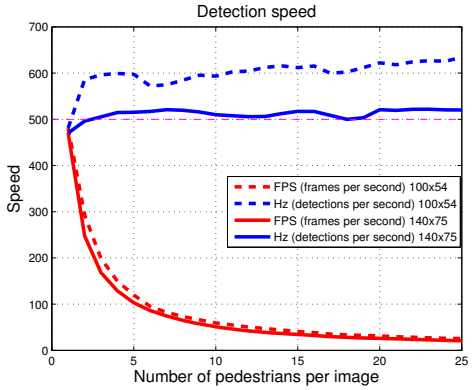


Figure 8. Speed results of our Warp Speed Pedestrian Detector for different ROI sizes.

cusses on image content with high probability of containing pedestrians at a fixed scale, thus being very fast. The speed of our detector evidently depends on the size of the ROI(s) we extract from the source image. This influence is given in figure 8, where we give the speed of our detector for two ROI sizes[1]. We display both the framerate (FPS) and the number of pedestrian detections per second (Hz) we can achieve with respect to the number of pedestrians in the image. Initially with only one pedestrian in the image we achieve 480 FPS, using the 140 pixels high ROI as motivated in section 4. If there are more pedestrians per image our pipeline is used more efficiently, thus the number of detections per second increases. However, this evidently leads to a decrease in the number of frames per second. At *e.g*. 20 pedestrians per image we still achieve an impressive framerate of 25 FPS.

6. Conclusion & Future Work

We have presented the combination a fast hybrid CPU/GPU implementation and the exploitation of scene constraints, resulting in our WSPD. Using our hybrid detection algorithm while reducing the searching space allows pedestrian detection at 500 detections per second. Experiments concerning both accuracy and speed on the Caltech dataset show that this speed-up does not lead to a decrease in accuracy. We proposed a challenging real-life use-case for our WSPD: detecting vulnerable road users in the blind spot camera of trucks. In the future we plan to modify the hybrid detector to allow more flexible hardware configurations, since currently we depend on the presence of CUDA-capable GPUs for the feature pyramid calculation and multicore CPUs for model evaluation. Furthermore, we seek to develop a GPU implementation of the model evaluation part, such that both feature pyramid and model evaluation can be executed on both CPU and GPU, allowing the use of our detector in an arbitrary manner on any combination of CPUs and GPUs.

[1] Both warping and detection times are taken into account here.

References

[1] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Fast stixel computation for fast pedestrian detection. In *Proc. of ECCV*, pages 11–20, 2012.

[2] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *Proc of. CVPR*, pages 2903–2910, 2012.

[3] R. Benenson, M. Mathias, T. Tuytelaars, and L. Van Gool. Seeking the strongest rigid detector. In *CVPR*, 2013.

[4] H. Cho, P. Rybski, A. Bar-Hillel, and W. Zhang. Real-time pedestrian detection with deformable part models. In *IEEE Intelligent Vehicles Symposium*, August 2012.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of CVPR*, volume 2, pages 886–893, June 2005.

[6] F. De Smedt, L. Struyf, S. Beckers, J. Vennekens, G. De Samblanx, and T. Goedemé. Is the game worth the candle? Evaluation of OpenCL for object detection algorithm optimization. In *Proc. of PECCS*, 2012.

[7] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *Proc. of BMVC*, 2010.

[8] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *Proc. of BMVC*, 2009.

[9] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *Proc. of CVPR*, June 2009.

[10] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *In IEEE PAMI*, 99, 2011.

[11] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. *In IEEE PAMI*, 31(12):2179–2195, Dec. 2009.

[12] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/.

[13] P. F. Felzenszwalb, R. B. Girshick, and D. Mcallester. Cascade object detection with deformable part models. In *Proc. of CVPR*, 2010.

[14] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *In IEEE PAMI*, 32(9):1627–1645, 2010.

[15] R. E. Kalman et al. A new approach to linear filtering and prediction problems. *In Journal of basic Engineering*, 82(1):35–45, 1960.

[16] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *Proc. of CVPR*, pages 1–8, 2008.

[17] V. Prisacariu and I. Reid. fastHOG - a real-time gpu implementation of HOG. Technical report, Department of Engineering Science, Oxford University, 2009.

[18] K. Van Beeck, T. Tuytelaars, and T. Goedemé. A warping window approach to real-time vision-based pedestrian detection in a truck's blind spot zone. In *Proc. of ICINCO*, 2012.

[19] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. of CVPR*, volume 1, pages 511–518, 2001.