Human Detection Based on Large Feature Sets Using Graphics Processing Units

William Robson Schwartz
Institute of Computing, University of Campinas, Campinas-SP, Brazil, 13083-852
schwartz@ic.unicamp.br
`http://www.liv.ic.unicamp.br/~wschwartz`

Keywords: Human Detection, Large Feature Sets, Graphics Processing Unit, Partial Least Squares

Received:

To obtain a better representation for human detection, edge, color, and texture information have been combined and employed. However, this combination results in an extremely high-dimensional feature space. The large number of feature descriptors results in expensive feature extraction and requires a dimension reduction process. Frameworks based on general purpose graphics processing unit (GPU) programming have been successfully applied in computer vision problems and in this work we model the human detection problem so that multi-core CPUs and multiple GPU devices can be used to speed-up the process. The experimental results show significant reduction on computational time when compared to the serial CPU based approach.

Povzetek:

1 Introduction

Human detection is a topic of interest in computer vision since people's locations play an important role in applications such as human-computer interaction and surveillance. However, detecting humans in single images is a challenging task due to both inter- and intra-person occlusion and variations in illumination and individual's appearances and poses.

There are two main approaches to human detection: part-based [6, 8, 15] and subwindow-based [1,2,16]. The first class of methods consists of a generative process where parts of the human body are combined according to a prior model. The latter class of methods aim at effectively deciding if a human is located in a window by combining low-level features extracted from subwindows (or blocks).

The work of Dalal and Triggs [2] obtained high detection rates employing histograms of oriented gradients (HOG) as feature descriptors. Subsequent improvements in human detection have been achieved mostly by using combinations of low-level descriptors [7, 9, 16, 18]. A strong set of features provides high discriminatory power.

Edge, color and texture information are among the characteristics able to distinguish between humans and background [13]. These clues can be captured by low-level descriptors: the original HOG descriptors with additional color information, called *color frequency*, and features computed from co-occurrence matrices.

To allow more location and pose flexibility within the detection window and to capture information at different scales, features are extracted at different sizes, using overlapping blocks.

In order to augment the information of edge-based features, we combine the original HOG with features providing texture and color information. Texture is measured using classical co-occurrence matrices [4], which have been used previously for texture classification. To capture color information we use a simple color extension of HOG features, called color frequency. A consequence of augmenting the HOG features with color and texture features is an extremely high-dimensional feature space.

Even though good results can be achieved using the described feature combination (as we will show in the experiments), the computational cost is directly influenced by the large number of features considered; therefore, feature extraction and dimensionality reduction become very expensive processes and need to be optimized.

GPU devices have become a powerful computational hardware for a given price and multiple of such devices may be attached to a single computer. This results in a powerful computational tool when the application can be split in several independent parts suitable to run in parallel. The subwindow-based approach for human detection is suitable for GPU implementation since the detection windows for different regions in the image are independent and therefore can be considered in parallel. Therefore, this work models the human detection problem so that multi-core CPUs and multiple GPU devices can be used to speed-up the process.

This paper is organized as follows. Section 2 describes works related to the proposed one. Section 3 describes the serial approach for the problem of human detection. Then, in Section 4 we present the proposed parallel approach for the problem. Experimental results comparing the serial and parallel approaches are shown in Section 5. Finally, Section 6 concludes with some final remarks.

2 Related Work

Frameworks based on general purpose GPU programming have been applied in computer vision problems and have provided high performance computation in problems such as feature tracking and matching [14], real-time stereo [5, 19], background subtraction [3], and motion detection [20].

To speed-up the detection process, we design the human detector implementation in such a way that we are able to exploit parallel computational power provided by multi-core CPUs and GPU devices. Our design also avoids redundant computation of features shared by different detection windows. As demonstrated in the experimental results, the proposed approach leads to a significant reduction on the computational time.

The work developed by Zhang and Nevatia [21] also uses GPU for human detection. However, they do not exploit the computational power of the multiple cores available in nowadays CPUs and the availability of multiple GPU devices attached to the computer, as we propose in this work. In addition, since Zhang and Nevatia implement the work proposed in [2], only features based on HOG are implemented, which leads to poor results (as it will be shown in the experimental results).

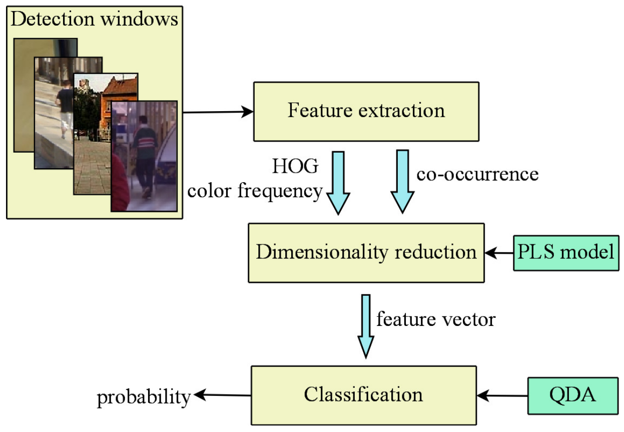


Figure 1: Serial approach. For each detection window in the image, features extracted using feature combination concatenated and analyzed by the partial least squares (PLS) model to reduce dimensionality, resulting in a low dimensional vector. Then, quadratic discriminative analysis (QDA) is used to classify this vector as either human or non-human. Finally, the probability of the detection window be classified as a human is output.

3 Serial Approach

In this section we review the serial approach for human detection, based on the work of Schwartz et al. [13]. The diagram shown in Figure 1 illustrates the steps of the serial approach for human detection. We decompose a detection window, d_i, into overlapping blocks and, extracting a set of features for each block, we construct the feature vector $\boldsymbol{v_i}$, describing d_i.

3.1 Feature Extraction

To capture texture, features extracted from co-occurrence matrices are used [4]. These matrices provide information regarding homogeneity and directionality of patches, which are important in human detection once a person tends to wear clothing composed of homogeneous textured regions and there is a significant difference between the regularity of clothing texture and background textures.

Edge information is captured using histogram of oriented gradients, which captures gradient structures that are characteristic of local shape [2]. In HOG, the orientation of the gradient for a pixel is chosen from the color band corresponding to the highest gradient magnitude. Some color information is captured by the number of times that each color band is chosen. A three bin histogram is build to tabulate the number of times that each color band is chosen. This feature is called color frequency [13].

3.2 Dimensionality Reduction

To handle the high dimensionality resulting from the combination of features, a statistical method called partial least squares (PLS) [17] is employed as a linear dimensionality reduction technique. PLS provides dimensionality reduction for even hundreds of thousands of variables, accounting for class labels in the process.

The basic idea of PLS is to construct a set of projection vectors $W = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots \boldsymbol{w}_p\}$ given the standardized data summarized in the matrix $\boldsymbol{X}$ of descriptor variables (features) and the vector $\boldsymbol{y}$ of response variables (class labels). The objective of the procedure is to derive a small, relevant set of latent variable vectors that captures the information inherent in the matrix $\boldsymbol{X}$ of descriptor variables in a compact form [12].

The dimensionality reduction is performed by projecting the feature vector $\boldsymbol{v_i}$ extracted from a detection window d_i onto the latent vectors $W = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots \boldsymbol{w}_p\}$. Vector $\boldsymbol{z_i}$ $(1 \times p)$ is obtained as a result. This vector is used in classification.

3.3 Classification

Once the feature extraction process is performed for all blocks inside a detection window d_i, features are concatenated creating an extremely high-dimensional feature vector $\boldsymbol{v_i}$. $\boldsymbol{v_i}$ is then projected onto set of latent variables W resulting in a low dimensional vector $\boldsymbol{z_i}$. For each vector $\boldsymbol{z_i}$, we use quadratic discriminant analysis to estimate probabilities for the two classes, human and non-human.

PLS tends to produce latent vectors that provide an approximately linear separation of the

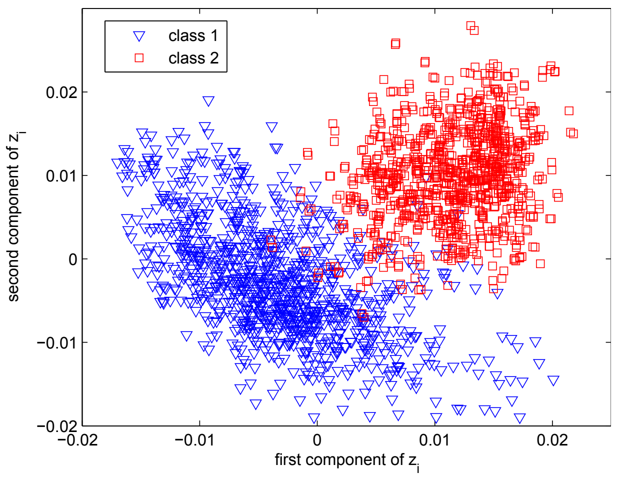


Figure 2: First two components of $\boldsymbol{z}_i$ obtained from projecting feature vectors into latent vectors for human and non-human classes. It is possible to see the clear separation between both classes provided by PLS used as a linear dimensionality reduction technique.

two classes. This enables us to use simple classifiers, such as QDA. Figure 2 shows the first two components of vectors $\boldsymbol{z_i}$ for the different classes extracted from the training data. We see that the classes do not overlap much, even in a 2-dimensional projection space. One of the advantages of using a simple classifier as QDA is that the computational time to perform the classification task is very low.

4 Parallel Approach

Designs like the one illustrated in Figure 1 are not able to take advantage of the full computational power provided by current computers. Therefore, it is of interest to design a detection approach that accomplishes the same task in a parallel fashion.

To exploit parallelism, we propose the approach illustrated in Figure 3. This design distributes the processing among the CPU and GPU devices available in the system to reduce the idle time of the processors. In the following subsections we describe the modules composing this approach.

4.1 Data Structure Creation

Before performing feature extraction, some data structures need to be created for each input image. Integral histograms are created for HOG and

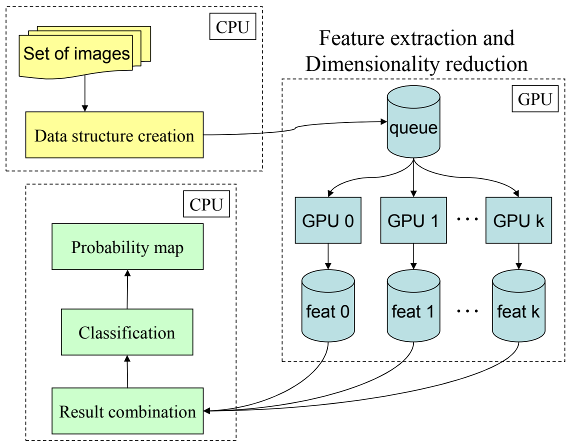


Figure 3: Parallel approach. Processing modules are executed simultaneously in CPU and GPU devices. Data structures used in co-occurrence and HOG methods are created in CPU. Such structures are used during the feature extraction and dimensionality reduction process performed on GPU. Finally, the low dimensional features resulting from multiple GPUs for each detection window are combined and classified.

matrices for the co-occurrence methods.

Once the time to create the data structures is substantially smaller than the feature extraction and dimensionality reduction (as we will show in the experiments), these two set of operations may be decoupled and executed in parallel increasing the use of the computational resources. The data structure creation is performed on the CPU and the feature extraction and dimensionality reduction on the GPU. To synchronize between these devices, we add a queue so that once the data structures are created for an image, they are stored in the queue until the GPU devices become available to process that particular image.

4.2 Feature Extraction and Dimensionality Reduction

In the serial approach, a detection window is decomposed into overlapping image blocks from which are extracted features to compose a feature vector. Since features for different image blocks can be extracted independently and a linear technique is used for dimensionality reduction, we can exploit the multiprocessors available in GPUs to extract features and reduce the dimensionality of multiple image blocks simultaneously, then, at the end, combine the results to obtain a low dimensional feature vector to describe a detection window.

In the GPU device, each multiprocessor consists of a set of scalar processor cores and employs an architecture called SIMT (single-instruction, multiple-thread). The multiprocessor maps each thread to one scalar processor core, and each scalar thread executes independently with its own instruction address and register state. The implementation in this work uses the parallel computing architecture developed by NVIDIA called Compute Unified Device Architecture (CUDA) [11]. The extensions to C programming language provided by CUDA allow that general-purpose computation be performed in GPUs in a well-defined and structured manner.

To exploit the set of multiprocessors in a GPU, several image blocks are processed simultaneously. This way, the input of each GPU is a range of image blocks that need to be processed and the intermediate results of feature extraction and dimensionality reduction are stored in an array $\boldsymbol{f}$ containing an entry for each detection window in the current image. $\boldsymbol{f}$ is indexed by the detection window index, e.g. $\boldsymbol{f}_i$ represents low dimensional features for the i-*th* detection window. At the end, the results obtained from all GPUs are combined so that the detection windows can be classified.

Algorithm 1 Steps performed for each GPU.

```
Input: set of blocks ${b_i, b_{i+1}, \ldots, b_j}$.
  launch simultaneously GPU processes to extract features and reduce dimensionality of all image blocks $b_k \in \{b_k : k = i, \ldots, j\}$
Output: array $\boldsymbol{f}$ containing low dimensional feature vectors for each detection window in the image.
```

Algorithm 1 describes the steps for each GPU device used during human detection. According to this algorithm, the feature extraction and dimensionality reduction are scalable with the number of GPU devices available since the image blocks can be divided among the devices and their processing is independent.

Algorithm 1 launches a multithread process to extract feature for each image block. Then, the dimensionality reduction for all detection windows sharing that specific block is performed. Once we use a sliding window to search for humans, one image block may be shared by several detection windows. The dimensionality reduction needs to be performed for each one of these detection windows. We describe this process as follows.

Given a image block b_k, features are extracted using co-occurrence, HOG and color frequency methods and a feature vector v_k is composed. Let $l_k = \{d_{k,0}, d_{k,1}, \ldots, d_{k,l}\}$ be the set of detection windows sharing the image block b_k. We project v_k onto latent vectors (learned using PLS) corresponding to each detection window $d_{k,j}$. This computes the contribution of features extracted from b_k to the detection window $d_{k,j}$. Finally, we add this contribution to $\boldsymbol{f}_{d_{k,i}}$, position corresponding to low dimensional feature vector for detection window $d_{k,j}$. Algorithm 2 shows the steps performed to process each image block assigned by algorithm 1.

Since the last step of Algorithm 2 adds the contribution of each image block to a detection window and multiple image blocks are processed simultaneously, two multithread processes may write at the same position of $\boldsymbol{f}$ at once. To prevent this behavior without incurring unwanted overhead, the image blocks are sorted so that different blocks being processed at the same time do not share a common detection window.

Algorithm 2 Steps to process each image block.

```
Input: image block $b_k$, list $l_k$ of detection windows sharing block $b_k$, projection vectors of PLS model for dimensionality reduction.
  $v_k \leftarrow$ co-occurrence, HOG and color frequency features extracted from image block $b_k$.
  for each detection window $d_{k,j} \in l_k$ do
    load projection vectors for block $b_k$ for detection window $d_{k,j}$
    project $v_k$ onto projection vectors
    add projection result to $\boldsymbol{f}_{d_{k,j}}$
  end for
```

4.3 Classification

Once the feature extraction and dimensionality reduction is finished for an image, classification is performed. This module is performed on the CPU because results (low dimensional feature vectors from GPUs) need to be combined prior to the classification. Additionally, due to the asynchronism between the CPU and GPU devices, queues store the low dimensional feature vectors outputted from each GPU device. After that, the feature vectors corresponding to an image are added and then the classification is performed resulting in a probability map for each image.

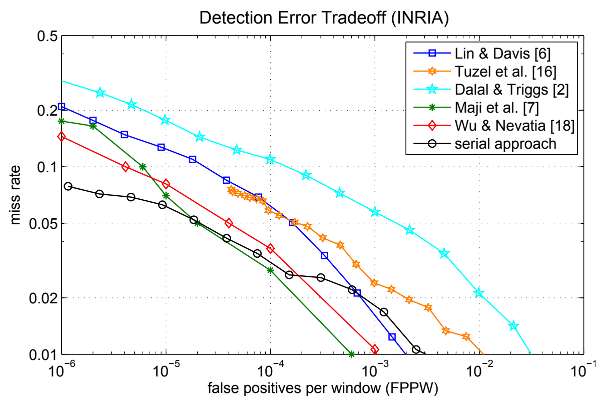
The addition of queues in the process also allows the use of heterogeneous GPU devices, e.g. GPU models presenting different computational power.

5 Experimental Results

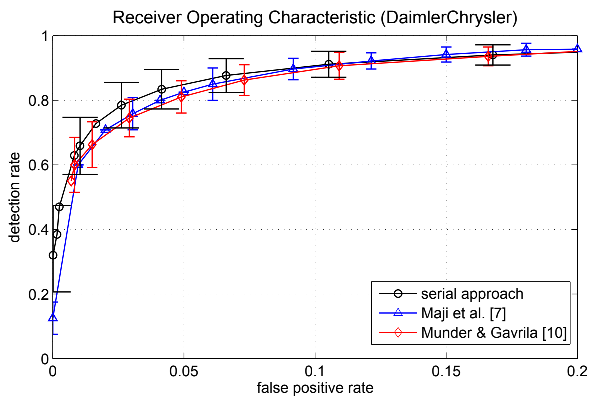
Implementation Details For the INRIA Person dataset [2], the setup of the feature extraction proposed in [13] is used, as described as follows. The co-occurrence features use block sizes of 16×16 and 32×32 with strides of 8 and 16 pixels respectively. The color space is converted to HSV and for each color band, 12 features are extracted from co-occurrence matrices created for each one of the four directions. The displacement considered is 1 pixel and each color band is quantized into 16 bins. The described setup results in $63,648$ features.

For HOG feature extraction blocks with sizes ranging from 12×12 to 64×128 are considered, resulting on $2,748$ blocks. For each block, 36 features are extracted, resulting in a total of $98,928$ features. In addition, the same set of blocks is employed to extract features using the color frequency method. This results in three features per block, and the total number of resulting features is $8,244$. Considering the aggregation of the three feature channels, the resulting feature vector extracted from one detection window contains $170,820$ elements.

Performance of the Serial Approach Initially, we describe results achieved by the serial detection method in order to show that efforts to obtain a fast implementation are worthwhile. We compared its performance to other methods in the literature using two standard human detection datasets: the INRIA Person dataset [2] and the DaimlerChrysler Pedestrian Classification Benchmark dataset [10]. For the DaimlerChrysler dataset, the number of elements in the feature vector is reduced due to he smaller size of



(a) INRIA person dataset



(b) DaimlerChrysler pedestrian dataset

Figure 4: Comparison of the serial approach used in the optimization with other methods in multiple human detection datasets.

the samples.

Figure 4(a) compares results obtained by the serial approach used in the GPU optimization with works published previously. Figure 4(b) compares results obtained by the serial approach with results reported in [7, 10]. Curves in Figure 4(a) are presented using detection error tradeoff curves on log-log scales. The x-axis corresponds to false positives per window (FPPW), defined by *FalsePos/(TrueNeg + FalsePos)* and y-axis shows the miss rate, defined by *FalseNeg/(FalseNeg + TruePos)*. While, curves in Figure 4(b) show detection rate instead of miss rate on the y-axis and both axes are shown in linear scales.

For both datasets, the results obtained with the described feature combination method outperform the other methods in regions of low false alarm rates, which are considered important regions for the human detection problem.

In addition, Figure 5 shows results obtained in full images obtained from the Google images and from the testing set of the INRIA Person dataset. To be able to detect people with different sizes, 11 different scales were considered for each image. The high number of detection windows resulting from the multiple-scale detection in each image also justifies the idea of considering a GPU-based implementation to speed-up the detection process.

Speed-up To test the performance of the parallel implementation we conducted experiments using sets of GPU devices NVIDIA GeForce 9800 GX2 and NVIDIA Tesla C870. Four GPU devices model GeForce 9800 GX2 were available in a Intel Core 2 Quad Q9450 2.66GHz with 4GB of RAM memory and two GPU devices model Tesla C870 were available in an AMD Opteron Dual 2218 2.6GHz with 2Gb of RAM memory. In the experiments we performed human detection in 100 images with 320×240 pixels using shift of 4 pixels between consecutive detection windows.

Table 1 shows the computational time obtained by the serial and parallel approaches using GeForce 9800 GX2 GPU devices. Similarly, Table 2 shows the results obtained using Tesla C870 GPU devices. Transfer time refers to the time spent copying data from the CPU to GPU, and vice-versa. Overhead is the computational time required to cache the features to be used for different detection windows in the serial approach. On the serial implementation, the cache is implemented so that features computed for an image block are stored to be used subsequently by different detection windows sharing that block.

According to Tables 1 and 2 we see that the parallelism between CPU and GPU is being exploited since the data structure creation time does not contribute significantly to the total computational cost. In addition, the multiprocessors in the GPU allow time reduction during feature extraction and dimensionality reduction.

The last two rows of tables 1 and 2 show the number of detection windows processed per second and the speed-up obtained when compared to the serial approach. Although the gain in speed is not linearly proportional to the number

	CPU	1 GPU	2 GPUs	3 GPUs	4 GPUs
Data structure creation	-	39.86s	41.82s	43.97s	47.42s
Transfer time	-	21.27s	23.96s	26.44s	34.83s
Overhead	243.76s	-	-	-	-
Feature extraction + dim. reduction	785.58s	394.06s	207.97s	137.56s	112.31s
Total time	1029.34s	418.14s	217.15s	149.19s	128.95s
detections/second	183.44	451.57	869.55	1265.65	1464.41
speed-up	1×	2.4×	4.7×	6.8×	7.9×

Table 1: Computational time and speed-up for serial and parallel approaches using NVIDIA GeForce 9800 GX2.

	CPU	1 GPU	2 GPUs
Data structure creation	-	66.06s	93.52s
Transfer time	-	8.20s	26.87s
Overhead	904.02s	-	-
Feature extraction + dim. reduction	1897.48s	228.68s	133.29s
Total time	2801.50s	238.02s	164.05s
detections/second	67.40	793.30	1151.01
speed-up	1×	11.7×	17.1×

Table 2: Computational time and speed-up for serial and parallel approaches using NVIDIA Tesla C870.

of GPUs, due to increasing time to data structure creation and data transfer, we see significant speed-up when more GPU devices are added into the system.

6 Conclusions

In this work we described a parallel design exploiting multi-core CPUs and multiple GPU devices for the human detection problem. The results have shown that the computational power of both CPU and GPUs can be exploited to obtain a faster implementation.

Even though the optimization framework described in this paper is focused on human detection, it is general enough to be easily applied to other object detection tasks relying on sliding detection windows.

Acknowledgments

This research was supported by FAPESP grant 2010/10618-3.

References

[1] J. Begard, N. Allezard, and P. Sayd. Real-Time Human Detection in Urban Scenes: Local Descriptors and Classifiers Selection with AdaBoost-like Algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[2] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Intl. Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[3] A. Griesser, S. De Roeck, A. Neubeck, and L. Van Gool. GPU-Based Foreground-Background Segmentation using an Extended Colinearity Criterion. In *Vision, Modeling, and Visualization*, pages 319–326, 2005.

[4] R. Haralick, K. Shanmugam, and I. Dinstein. Texture Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6), 1973.

[5] P. Labatut, R. Keriven, cole Nationale, and P. Chausses. A GPU Implementation of Level Set Multi-View Stereo. In *International Conference on Computational Science Workshop General Purpose Computation on Graphics Hardware*, pages 212–219, 2005.

[6] Z. Lin and L. S. Davis. A Pose-Invariant Descriptor for Human Detection and Segmentation. In *European Conference on Computer Vision*, pages 423–436, 2008.

[7] S. Maji, A. Berg, and J. Malik. Classification using Intersection Kernel Support Vector Machines is Efficient. In *IEEE Intl. Conference on Computer Vision and Pattern Recognition*, 2008.

[8] K. Mikolajczyk, C. Schmid, and A. Zisserman. Human detection based on a probabilistic assem-

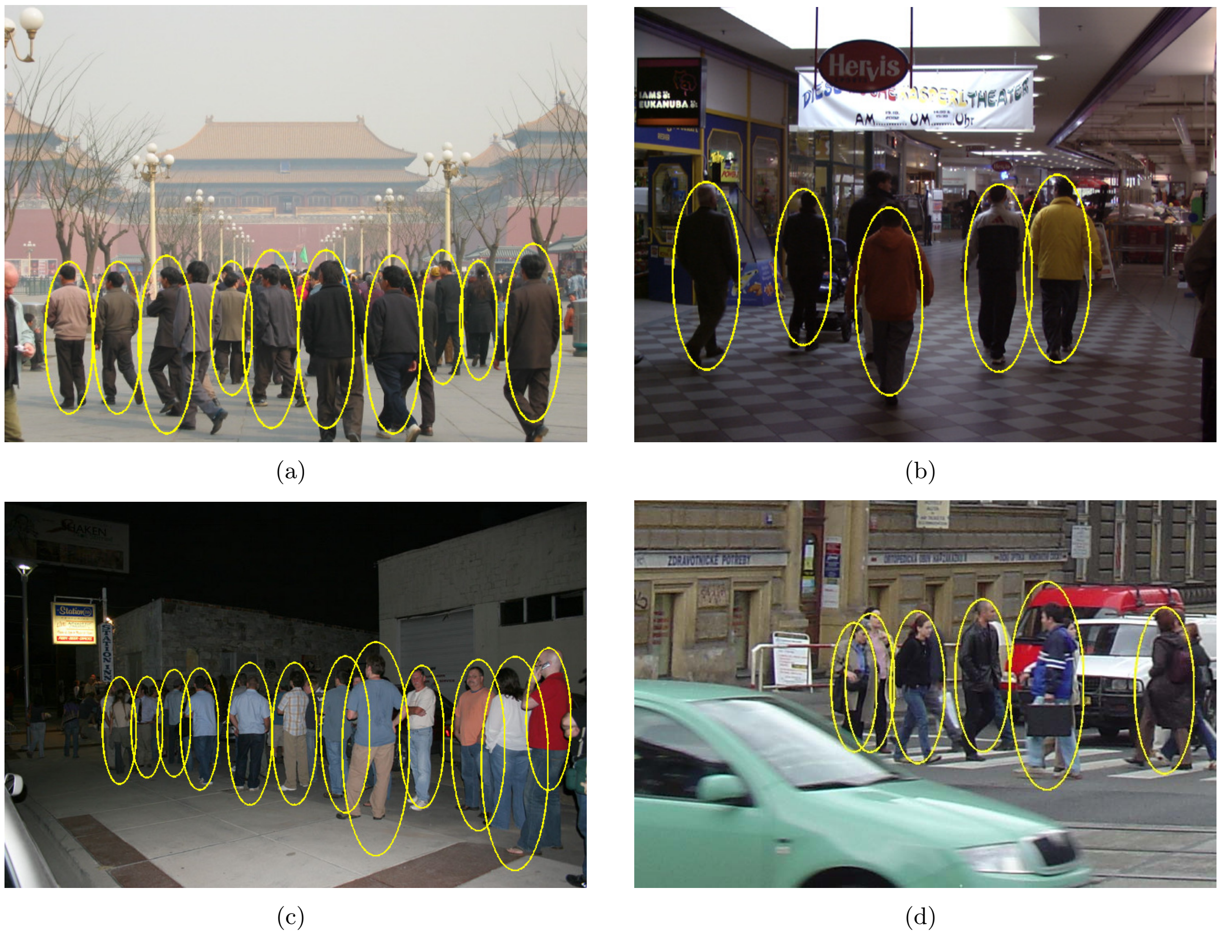


Figure 5: Results obtained in full images containing people of multiple sizes.

bly of robust part detectors. In *European Conference on Computer Vision*, 2004.

[9] Y. Mu, S. Yan, Y. Liu, T. Huang, and B. Zhou. Discriminative Local Binary Patterns for Human Detection in Personal Album. In *IEEE Intl. Conference on Computer Vision and Pattern Recognition*, 2008.

[10] S. Munder and D. Gavrila. An Experimental Study on Pedestrian Classification. *PAMI*, 28(11):1863–1868, 2006.

[11] NVIDIA. *NVIDIA CUDA Programming Guide 2.0*. 2008.

[12] R. Rosipal and N. Kramer. Overview and Recent Advances in Partial Least Squares. *Lecture Notes in Computer Science*, 3940:34–51, 2006.

[13] W. R. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis. Human Detection Using Partial Least Squares Analysis. In *IEEE International Conference on Computer Vision*, pages 24–31, 2009.

[14] S. N. Sinha, J. michael Frahm, M. Pollefeys, and Y. Genc. GPU-based Video Feature Tracking and Matching. Technical report, In Workshop on Edge Computing Using New Commodity Architectures, 2006.

[15] D. Tran and D. Forsyth. Configuration Estimates Improve Pedestrian Finding. In *NIPS 2007*, pages 1529–1536. 2008.

[16] O. Tuzel, F. Porikli, and P. Meer. Human Detection via Classification on Riemannian Manifolds. In *IEEE Intl. Conference on Computer Vision and Pattern Recognition*, 2007.

[17] H. Wold. Partial least squares. In S. Kotz and N. Johnson, editors, *Encyclopedia of Statistical Sciences*, volume 6, pages 581–591. 1985.

[18] B. Wu and R. Nevatia. Optimizing Discrimination-Efficiency Tradeoff in Integrating Heterogeneous Local Features for Object Detection. In *IEEE Intl. Conference on Computer Vision and Pattern Recognition*, 2008.

[19] R. Yang and M. Pollefeys. Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware. pages 211–217, 2003.

[20] Q. Yu and G. Medioni. A GPU-Based Implementation of Motion Detection from a Moving Platform. 2008.

[21] L. Zhang and R. Nevatia. Efficient Scan-Window Based Object Detection Using GPGPU. In *Computer Vision and Pattern Recognition Workshops*, 2008.