Efficient GPU Implementation of Informed-Filters for Fast Computation

Takuro Oki[1(✉)] and Ryusuke Miyamoto[2]

[1] Department of Computer Science, Graduate School of Science and Technology, Meiji University, Tokyo, Kanagawa 2148571, Japan
o_tkr@cs.meiji.ac.jp

[2] Department of Computer Science, School of Science and Technology, Meiji University, Tokyo, Kanagawa 2148571, Japan
miya@cs.meiji.ac.jp

Abstract. Human detection is an important task for several practical applications that require high-speed processing with good detection accuracy. This paper proposes a high-speed implementation of Informed-Filtersthat shows excellent accuracy in human detection. Our implementation reduces memory access during feature calculation and realizes efficient computation on an NVIDIA GPU where a thread is allocated to a detection sub-window. Experimental results using top-view images considering surveillance from UAVs showed that the processing speed was about 100 fps for 2560×1352 images on an NVIDIA 980Ti GPU, whereas it was 5.4 fps on an Intel Xeon 2.30 GHz CPU.

1 Introduction

Vital sensing during exercise on the basis of multi-hop sensor networks is useful for preventing sudden unwellness and for improving the effectiveness of training [1]. To realize such sensor networks, a novel routing scheme is indispensable because conventional schemes that use RSSI or GPS cannot account for the high density and moving speed of sensor nodes attached to humans doing exercise. The target applications of these sensor networks are not limited to only one kind of exercise but include several kinds of exercises as shown in Fig. 3. Therefore, vital sensor networks should work well for several node densities and moving speeds.

To enable the effective routing useful for such multi-hop networks, the authors are trying to realize Image Assisted Routing where the locations of sensor nodes are determined with visual information obtained from several cameras mounted on unmanned aerial vehicles (UAVs). An overview of the Image Assisted Routing is shown in Fig. 4. Remarkable advances in object detection and tracking enable accurate localization of humans wearing vital sensors; however, real-time processing of the localization on embedded systems has not been achieved yet. The huge computation requirements are an especially significant problem in the implementation of human detection on embedded systems.

© Springer International Publishing AG, part of Springer Nature 2018
M. Paul et al. (Eds.): PSIVT 2017, LNCS 10749, pp. 302–313, 2018.
https://doi.org/10.1007/978-3-319-75786-5_25

Several schemes aim to quickly compute object detection [2,3]. The authors are taking several approaches toward realizing accurate and high-speed object detection [4–8]. However, the classification accuracy of [2] is insufficient for human detection, and [3] requires stixel images obtained from a stereo camera that is not used in generic scene and especially in aerial images that are used in our project. Moreover, the classification accuracy of [3] is worse than Informed-Filters [9] that exhibit state-of-the-art classification accuracy in human detection. A novel approach [10,11] for object detection that does not use exhaustive search on the basis of sliding windows has been developed. [10,11] show good accuracy for multi-class object detection; however, the accuracy of one-class object detection for human target is not sufficient. Therefore, the authors are trying to speed-up Informed-Filters-based human detection by using an NVIDIA GPU.

GPU implementation of Informed-Filtersis more difficult than that of Intergral channel features [12] and Aggregate channel features [13] because feature computation of Informed-Filtersis more complex than that of intergral channel features and aggregate channel features, which use simple rectangular features as shown in Fig. 1; integral images can drastically reduce computation time in these schemes. In Informed-Filters, ternary features as shown in Fig. 2 are used to improve classification accuracy considering the complicated shapes of detection targets but feature computation using ternary features requires data to be accessed more frequently to generated integral images than intergral channel features and aggregate channel features. To solve this problem, a novel and parallel implementation that reduces the number of data access to pixels for feature computation and parallel implementation by using an NVIDIA GPU is shown. Our implementation reduces branch divergence and minimizes data transfer between a CPU and a GPU to achieve high speed processing without deteriorating of detection accuracy.

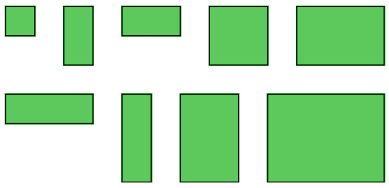
Fig. 1. Square features

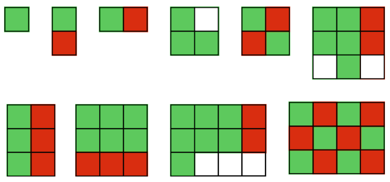
Fig. 2. Ternary features

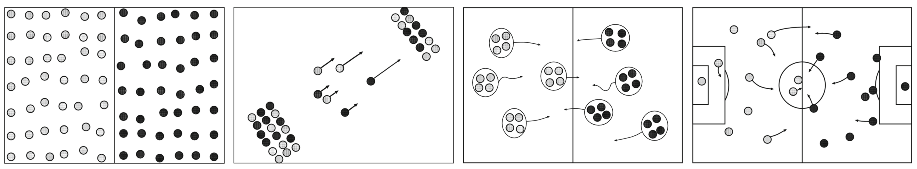
Fig. 3. Several kinds of motion patterns during exercise

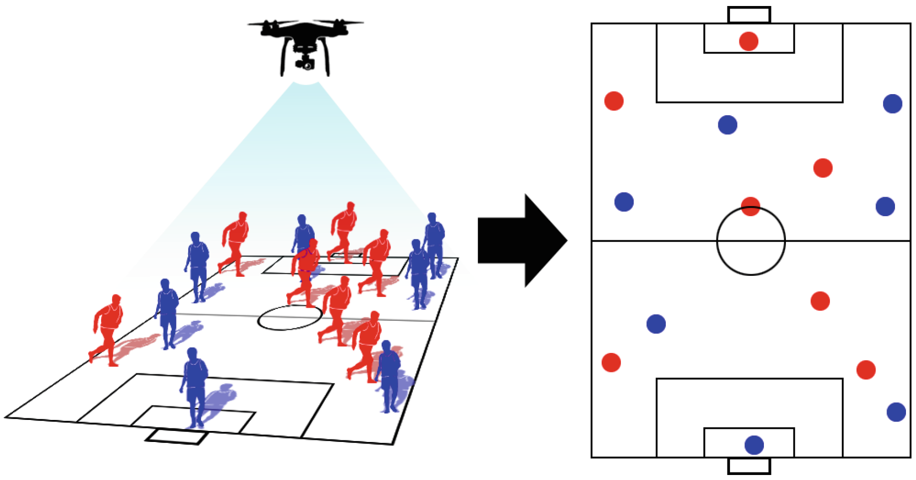

Fig. 4. Concept of image assisted routing

2 How to Reduce the Frequency of Data Access in the Informed-Filters

This section analyzes the processing flow of Informed-Filtersand shows how to reduce the frequency of data access in feature computation on the basis of an appropriate division of rectangular features that is much more efficient than intergral channel features and aggregate channel features.

2.1 The Frequency of Data Access During Feature Computation

In the Informed-Filters, a strong classifier is constructed with many decision trees from weak classifiers selected by boosting. These decision trees calculate the score for classification using rectangular features called filters as shown in Fig. 2. The types and locations of these filters composed of cells are determined at the training process by boosting. A cell is a unit rectangle and has a label assigned from $\{+1, -1, 0\}$. In the Informed-Filters, a feature F corresponding to a filter is computed by the following equation:

$$F = \sum_{i=0}^{n} L(cell_i) \cdot I(cell_i), \tag{1}$$

where n, $I(cell_i)$, and $L(cell_i)$ mean the number of cells included in the filter, the sum of pixels included in a $cell_i$, and a label assigned to a $cell_i$, respectively.

Equation (1) shows that feature computation requires frequent data access to pixels. A straightforward implementation of this computation needs $width \times height \times n$ read operations from pixels when the size of a cell is $width \times height$.

The number of read operations is still $4 \times n$ even if integral images are used in feature computation; this is about n times as large as the number of read operations used in intergral channel features or aggregate channel features.

2.2 Reduction of Read Operations in Feature Calculation by Filter Division

To reduce the number of read operations for feature calculation in Informed-Filters, the authors decided to divide a filter into several rectangles whose sizes are as large as possible: this division minimizes the number of divided regions. Using this division, the sizes of rectangles are larger than the sizes of cells, and the number of read operations can be reduced. In the proposed dividing scheme, first cells included in a filter are merged to a composite rectangle according to their labels, and then the generated composite rectangle is divided into several rectangles while minimizing the number of divisions. In the proposed scheme, this division is performed by [14], whose procedure can be summarized as follows:

Phase 1. Obtain degenerate chords from a composite rectangle and divide the composite rectangle according to the degenerate chords,
Phase 2. store vertices in a divided region clockwise,
Phase 3. scan the stored vertices and sort all horizontal line segments in descending order about $y-$coordinate, and
Phase 4. divide a composite rectangle by the most appropriate pattern selected from Fig. 6.

These operations are detailed in the rest of this subsection.

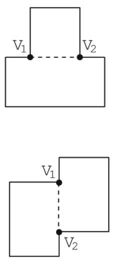

Fig. 5. Degenerate chords

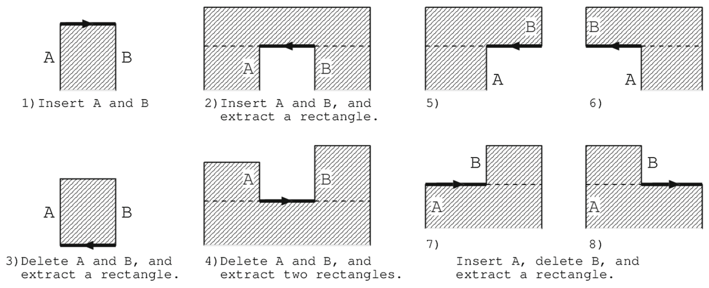
A
B
1) Insert A and B
A
B
2) Insert A and B, and extract a rectangle.
B
A
5)
B
A
6)
A
B
3) Delete A and B, and extract a rectangle.
A
B
4) Delete A and B, and extract two rectangles.
B
A
7)
B
A
8)
Insert A, delete B, and extract a rectangle.

Fig. 6. Pattern

Phase 1. This operation obtains degenerate chords that minimize the number of divisions from a composite rectangle. A degenerate chord is a line segment included in a composite rectangle whose both endpoints are recessed points having the same $x-$coordinate or $y-$coordinate. Figure 5 shows examples of degenerate chords as dashed lines where V_1 and V_2 are recessed points. If a composite rectangle has any degenerate chords, the composite rectangle is divided into several rectangles with no degenerate chords. After this division, a minimum division of the composite rectangle can be obtained by a minimum division of the rectangles generated by this division. In this operation, appropriate degenerate chords must be selected because a minimum division of the composite rectangle is not obtained if the selected degenerate chords are dependent as shown in [14].

A problem in obtaining a maximum independent set of degenerate chords can be represented by a bipartite graph $G(V_d, E)$ that is composed of line segments generated from endpoints of overlapping degenerate chords. V_d and E show a set of degenerate chords and line segments generated from endpoints of overlapping degenerate chords, respectively. A maximum independent set of vertices V_d' shows a maximum set that does not have any common edges among vertices included in the set. Therefore, the maximum set is equal to a maximum set of dependent degenerate chords. A maximum independent set of vertices can be computed by the following equation(Gallai) using a minimum vertex cover $MinV_{cover}$ obtained from maximum matching M' of a bipartite graph G(König)

$$V_d' = V_d - MinV_{cover} \tag{2}$$

Phase 2. In this operation, vertices included in a composite rectangular are scanned as internal regions and are located at the left side of the edge between the current and the next vertices. This scan order is represented by the word "clockwise".

Phase 3. In this operation, a scanning direction for a line segment is stored. Sorted horizontal line segments are stored in YList according to $y-$coordinate, and vertical line segments are stored in XList whose initial state is empty according to $x-$coordinate.

Phase 4. The following operations are applied to a line segment H obtained from YList. One pattern is selected from eight patterns shown in Fig. 6 considering the direction of the line segment H and the number of line segments included in XList that includes both endpoints of H. After this classification, some operations corresponding to the selected type are executed as shown in Fig. 6, and division is applied to a composite rectangle if possible. The above operations are applied to all horizontal line segments.

3 GPU Implementation of the Informed-Filters

This section explains how to apply parallel processing to Informed-Filtersusing a GPU, the effect of the soft cascade structure, and the improvement of memory access for high-speed processing.

3.1 Parallel Implementation Strategy on a GPU

Both search windows and weak classifiers can be computed in parallel in the case of detection by exhaustive search on the basis of sliding windows: parallel processing based on multiple weak classifiers has higher parallelism than multiple search windows. However, we cannot focus only on parallelism that is expected to perform well because the frequency of memory access and branch divergence that is caused when different branch operations are required in a warp are more important than parallelism itself in an implementation using CUDA [15] with an NVIDIA GPU.

If multiple weak classifiers of a strong classifier constructed by Informed-Filtersare computed in parallel, a block that is assigned to a streaming multiprocessor in a GPU handles a sub-window extracted from an input image, and each thread that is a smaller computational unit included in a block computes features independently using a weak classifier as shown in Fig. 7. In this case, branch divergence occurs frequently because a branch direction of a weak classifier depends on input data and on branch directions being generally different from each other.

Consequently, the authors use a parallel implementation for search windows as shown in Fig. 8, where a thread is allocated to the computation of a search window. In this implementation, the computation amount allocated to a thread is larger, but there is no branch divergence caused by the computation of a weak classifier composed of a decision tree. To improve the computation speed, the efficient scheme for feature calculation proposed in the previous section is used in addition to the window-based parallel implementation.

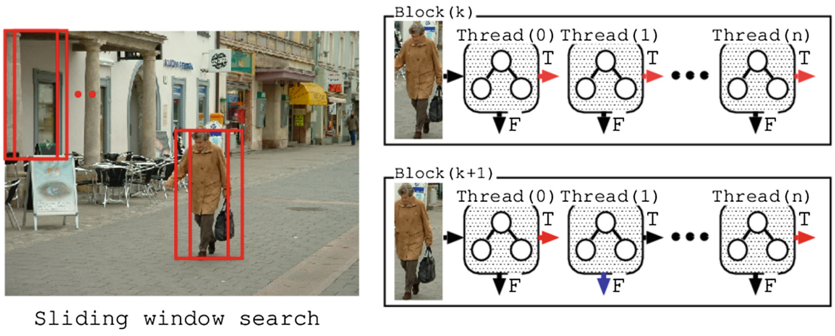

Fig. 7. Block per window

3.2 Effect of the Soft Cascade Structure

A strong classifier used in this research has a soft cascade structure as shown in Fig. 9. This structure enables fast computation without deterioration in detection accuracy: an input sub-window that does not obviously include a detection

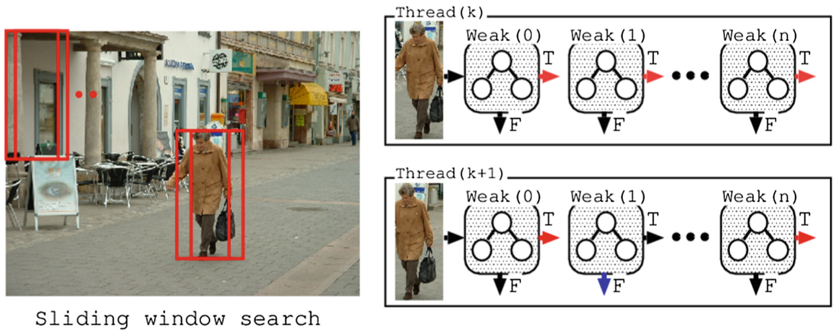

Fig. 8. Thread per window

target is rejected at an early stage when the accumulated score given by each weak classifier becomes lower than a threshold that is determined prior during the training process. Our implementation may cause branch divergence because some sub-windows are rejected in early stages, but others should be checked by the subsequent stages. Here the branch divergence means a rejection of an input sub-window, and the computation of the thread becomes unnecessary, but the efficiency of parallel execution becomes lower. However, the effect of early rejection by the soft cascade structure is powerful for object detection by a sliding window search because the number of sub-windows not including a target object is much larger than the number of sub-windows including a target object. Therefore, the soft cascade structure of a strong classifier is used in our implementation.

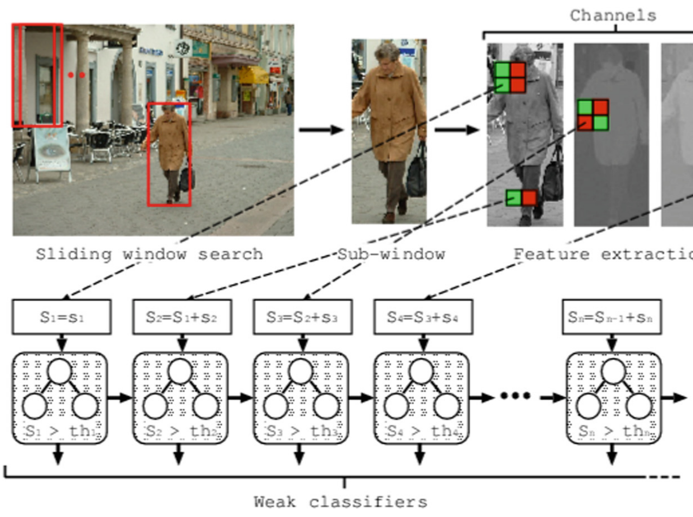

Fig. 9. Softcascade structure

3.3 Memory Access Improvement by the Constant Memory

The implementation described above executes feature computation using the same weak classifier at different threads that are computed in parallel. This operation causes many simultaneous requests to data corresponding to a weak classifier that may decrease the computation speed. To solve this problem, data that retain weak classifiers are stored to the constant memory, which is specially designed cache memory: the constant cache enables high-speed simultaneous accesses from multiple threads. In addition, our implementation is carefully designed to reduce the memory access by using broadcasting of the constant memory that enables simultaneous data transfer between memory and threads in half of a warp: the frequency of data transfer can be reduced to $\frac{1}{16}$ in the best case scenario.

3.4 Efficient Data Transfer Between a CPU and a GPU

Data transfer between a host CPU and a GPU requires huge overhead. Therefore, this kind of operation should be reduced for fast computation. A simple implementation causes images to be copied several times from a CPU to a GPU owing to the multiple channel images used in Informed-Filters. To avoid the increase in data transferred between a CPU and a GPU, only a feature image as shown in Fig. 10 that is generated from multiple channel images is transferred per input image in our implementation.

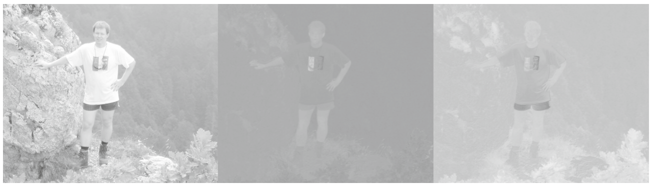

Fig. 10. Multiple channel images

4 Evaluation

This section evaluates the detection accuracy and the processing speed of the proposed implementation.

4.1 Experimental Conditions

In the evaluation, images as shown in Fig. 11 whose resolution were 2560×1352 were used. In the test images, several humans are located on the field and move appropriately. These images are created using sophisticated three dimensional computer graphics with human models and a field model to generate arbitrary views, where the human positions are determined by actual data obtained from an actual sports scene. Top view images were used because occlusions by other targets can be easily reduced in our project.

To construct a strong classifier, 200 weak classifiers composed of a decision tree whose depth were one was selected using 884 types of filters by Informed-Filters.

The specifications of the computer used to measure the processing speed is shown in Table 1.

Table 1. Computer specifications.

OS	Linux(Ubuntu 16.04)
CPU	Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30 GHz
Memory	128 GB, DDR4
GPU	GeForce GTX 980 Ti

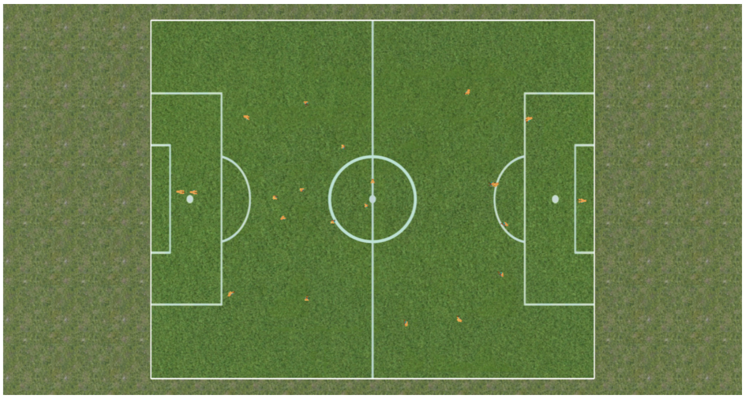

Fig. 11. Test data.

4.2 Detection Accuracy

Figure 12 shows the Detection Error Trade-off (DET) curves plotted using a miss rate and a false positive per image. These results show that the parallel implementation on a GPU can detect target humans the same as the sequential implementation on a CPU.

4.3 Processing Speed

To evaluate the processing speed, the average time consumed for the detection of an image was measured using 3000 test images. Table 2 shows the average processing speed by the parallel implementation on a GPU and the CPU implementation. These results show that the processing speed was about 18.58 times faster if parallel implementation on a GPU was applied. Then, Figs. 13 and 14 show the actual processing time for each test image. According to these results, the processing speed is stably faster if input images are changed.

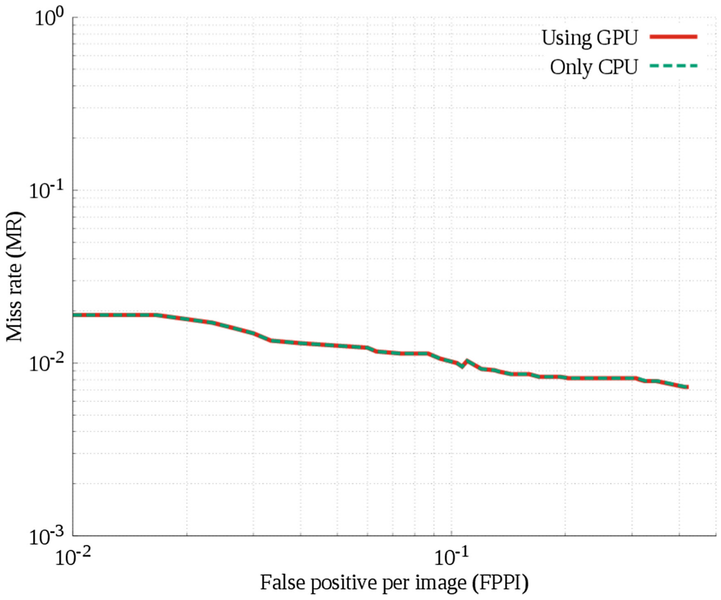

Fig. 12. DET curves.

Table 2. Processing speed.

	GPU	Only CPU
Average time	10.46 ms	186.54 ms

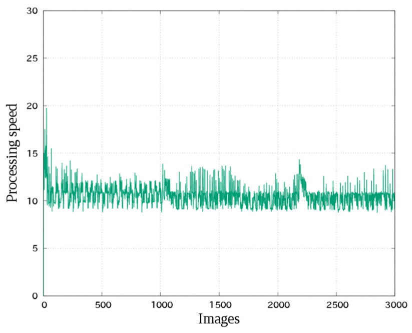

Fig. 13. Execution time for 3000 images by a GPU.

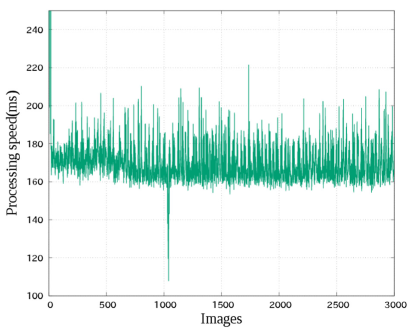

Fig. 14. Execution time for 3000 images by a CPU.

5 Conclusion

This paper proposed the parallel implementation of Informed-Filterson a GPU in order to realize real-time and accurate detection of humans in top view images captured from a camera mounted on a UAV. In our implementation, first computational costs caused by the huge number of read operations required for feature computation using filters composed of cells are reduced. To achieve this reduction, a minimum division for a composite rectangle is applied for filters used in the feature computation. Using this modification, the frequency of memory access to pixels was about 50% that of the straightforward implementation. In the GPU implementation, the authors decide to allocated a thread to a sub-window to reduce branch divergence during feature calculation; this approach seemed more suitable than others when the soft cascade structure was applied with the parallel implementation. In addition to the parallel implementation, appropriate use of the constant memory and a reduction in data transfer between a CPU and a GPU were applied in our implementation. Consequently, the processing speed of human detection in 2560×1532 top view images considering surveillance from UAVs was about 100 frames per second, whereas sequential implementation on a CPU processed at only about 5.4 frames per second.

Acknowledgment. The research results have been achieved thanks to "Research and development of Innovative Network Technologies to Create the Future", the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN.

References

1. Hara, S., Yomo, H., Miyamoto, R., Kawamoto, Y., Okuhata, H., Kawabata, T., Nakamura, H.: Challenges in real-time vital signs monitoring for persons during exercises. Int. J. Wirel. Inf. Netw. **24**(2), 91–108 (2017)
2. Oro, D., Fernández, C., Rodríguez, S.J., Martorell, X., Hernando, J.: Real-time GPU-based face detection in HD video sequences. In: Proceedings of IEEE International Conference on Computer Vision, pp. 530–537 (2011)
3. Benenson, R., Mathias, M., Timofte, R., Van Gool, L.J.: Pedestrian detection at 100 frames per second. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, pp. 2903–2910 (2012)
4. Miyamoto, R., Oki, T.: Soccer player detection with only color features selected using informed Haar-like features. In: Blanc-Talon, J., Distante, C., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2016. LNCS, vol. 10016, pp. 238–249. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48680-2_22
5. Hiromoto, M., Sugano, H., Miyamoto, R.: Partially parallel architecture for AdaBoost-based detection with Haar-like features. Proc. IEEE Trans. Circ. Syst. Video Technol. **19**, 41–52 (2009)
6. Hiromoto, M., Miyamoto, R.: Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features. In: Proceedings of IEEE International Conference on Computer Vision Workshops, pp. 894–899 (2009)

7. Hiromoto, M., Miyamoto, R.: Cascade classifier using divided CoHOG features for rapid Pedestrian detection. In: Fritz, M., Schiele, B., Piater, J.H. (eds.) ICVS 2009. LNCS, vol. 5815, pp. 53–62. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04667-4_6
8. Yu, J., Miyamoto, R., Onoye, T.: Fast pedestrian detection using a soft-cascade of the CoHOG-based classier: how to speed-up SVM classiers based on multiple-instance pruning. IEEE Trans. Image Process. **22**, 4752–4761 (2013)
9. Zhang, S., Benenson, R., Schiele, B.: Filtered channel features for Pedestrian detection. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, pp. 1751–1760 (2015)
10. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, pp. 779–788 (2016)
11. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, pp. 6517–6525 (2017)
12. Dollár, P., Tu, Z., Perona, P., Belongie, S.J.: Integral channel features. In: Proceedings of British Machine Vision Conference, pp. 1–11 (2009)
13. Nam, W., Dollár, P., Joon, H.H.: Local decorrelation for improved pedestrian detection. In: Proceedings of Advances in Neural Information Processing Systems, pp. 424–432 (2014)
14. Ohtsuki, T., Sato, M., Tachibana, M., Torii, S.: Minimum partitioning of rectilinear regions. IPSJ J. **24**, 647–653 (1983)
15. NVIDIA Corporation: nVidia CUDA Programming Guide. http://docs.nvidia.com/cuda/