Human Detection using Mobile Embedded Smart Cameras

Li He
University of Nebraska-Lincoln, heli@huskers.unl.edu

Youlu Wang
University of Nebraska-Lincoln, youlu.wang@huskers.unl.edu

Senem Velipasalar
University of Nebraska-Lincoln, velipasa@engr.unl.edu

M. Cenk Gursoy
University of Nebraska-Lincoln, gursoy@engr.unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/electricalengineeringfacpub
Part of the Electrical and Computer Engineering Commons

He, Li; Wang, Youlu; Velipasalar, Senem; and Cenk Gursoy, M., "Human Detection using Mobile Embedded Smart Cameras" (2011). Faculty Publications from the Department of Electrical and Computer Engineering, 206.
http://digitalcommons.unl.edu/electricalengineeringfacpub/206

This Article is brought to you for free and open access by the Electrical & Computer Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Faculty Publications from the Department of Electrical and Computer Engineering by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
Abstract—Embedded smart cameras are stand-alone units that combine sensing and processing on a single embedded platform. They allow flexibility in camera placement, and provide mobility without being dependent on wired links. On the other hand, detecting and tracking objects from videos captured by mobile cameras is a very challenging task even on powerful computers. Since embedded smart cameras have very limited processing power, memory and energy, the challenge becomes even bigger. In this paper, we present a person detection system using an embedded smart camera mounted on a remote-controlled car. We employ histogram of oriented gradients (HOG) for detection, and present the performance results obtained on these resource-constrained environments. The example application is patrolling hallways in a building to detect people.

I. INTRODUCTION

An embedded smart camera not only captures images, but also includes a processor, memory and communication interface making it a stand-alone unit. Battery-powered embedded smart cameras remove the dependency on wired links, and provide mobility and great flexibility in camera placement. Common computing platforms for smart cameras are FPGAs, digital signal processors (DSPs), and/or general purpose microprocessors [1].

Yet, embedded smart cameras have limited processing power, memory and energy. A computer vision task, which is challenging to accomplish even on powerful computing platforms, becomes much more challenging when it needs to be implemented on the resource-limited embedded cameras.

Object detection and tracking are interesting yet challenging problems even with static cameras. The difficulties include non-static backgrounds, variability of objects, crowded scenes, occlusions and changes in the environment, such as illumination changes. Many foreground object detection methods have been introduced in literature that are based on background subtraction [2], [3], [4]. Most of these methods build and update a background model. Kim et al. [4] proposed a codebook-based algorithm for background modeling. Sample background values at each pixel are quantized into codebooks during the training stage, representing a compressed form of the background model. Wang et al. [5] use a lightweight background subtraction algorithm described in [3]. Compared to state-of-the-art background subtraction algorithms, the memory requirement of the algorithm described in [3] is very small making it suitable for embedded platforms.

As stated above, most of the background subtraction methods build a model for the background, and if the background model is not properly updated, the performance can degrade due to shadows and illumination changes which are common in real-world scenarios. Moreover, they assume that the cameras are static, which restricts the possible application areas.

Detecting and tracking moving objects with mobile cameras is a very challenging task due to various uncertainty in object locations, measurement noise, clutter, changing background and occlusions. Since the background changes continuously, background subtraction methods are inapplicable in these scenarios. Object detection algorithms have been introduced more recently, and they employ an object detector instead of background subtraction. They take a single image as input and output the regions where the objects of interest are located. Thus, they can be used with moving cameras.

Dalal and Triggs [6] presented a descriptor for human detection that is based on Histogram of Oriented Gradients (HOG). The basic idea of HOG is that local visual features can be characterized well by the distribution of local intensity gradients or edge directions. An image is divided into cells and 1-D histograms of gradient directions are calculated. The calculated histograms are then normalized over cells to get the HOG. A support vector machine (SVM) is then trained to classify human and non-human regions.

The HOG descriptor has been used for foreground detection, instead of conventional background subtraction algorithms, in the tracking algorithms in recent years. Bilinski et al. [7] used HOG descriptor to track the feature points on the objects to handle the occlusions. Sugano and Miyamoto [8] perform pedestrian tracking using particle filter and multiple features including HOG and color histogram on an NVIDIA GPU.

Combining the HOG detector and a conventional tracking algorithm is a recent trend for object tracking. Breitenstein et al. [9], [10] propose a tracking-by-detection method. HOG-based human detector is used to detect humans in video sequences and the detection outputs are then used as inputs to a particle filter-based tracking algorithm. Although detection
algorithms and tracking-by-detection framework are more suitable for moving cameras and more robust towards background changes, they have only been implemented on powerful PC platforms, and the processing speed is not too fast even on these platforms [10].

Two of the biggest advantages of the embedded smart cameras are local processing capability and mobility. Thus target detection needs to be performed on the embedded platforms as the next step of having operational mobile embedded smart cameras. When we want to implement the detection algorithms on these resource-constrained environments, the challenge becomes even bigger.

In this paper, we implement the HOG-based human detector on embedded smart cameras and test its performance with static and mobile cameras for different scenarios. We mounted cameras on remote-controlled cars, and report results and processing times.

Ability to detect objects with moving cameras has application in different areas including robotics, surveillance and smart driving systems. The example application we chose here is patrolling hallways for detecting people. The presented results are very promising, and provide insight on the capabilities and limitations of these embedded platforms.

The rest of the paper is organized as follows: The wireless embedded smart camera platform employed in our experiments is described in Section II. The human detection algorithm using HOG descriptors and SVM is summarized in Section III. The experimental results are presented in Section IV, and the paper is concluded in Section V.

II. HARDWARE PLATFORM

The embedded smart camera employed in this work is a CITRIC mote [11] seen in Fig. 1. The camera board is composed of an image sensor, a fixed-point microprocessor, external memories and other supporting circuits. The camera is capable of operating at 15 frames per second (fps) in VGA and lower resolutions.

The image sensor of the camera board is a Omni Vision OV9655, which is a low voltage SXGA CMOS image sensor and designed to perform well in low-light conditions. It supports image sizes SXGA ($1280 \times 1024$), VGA ($640 \times 480$), and any size scaling down from VGA. The microprocessor PXA270 is a fixed-point processor with a maximum speed of 624MHz and 256KB of internal SRAM. It is capable of working in low voltage and low frequency, as low as 0.85V and 13MHz, to achieve low power consumption. Besides the internal memory of the microprocessor, the PXA270 is connected to 64MB of SDRAM and 16MB of NOR FLASH. A USB-to-UART bridge controller is connected between the PXA270 UART port and the USB port on a personal computer. The camera board can be powered by the USB port from a personal computer, or four AA batteries.

We have run all of our experiments in QVGA ($320 \times 240$) resolution. The algorithms run on the embedded Linux system imported onto the microprocessor. The frames of interest can be saved in JPEG format on the SDRAM.

III. HUMAN DETECTION USING HOG AND SVM

A. HOG descriptor

The HOG descriptor models the local object texture by calculating the distribution of the local intensity gradients and the edge directions [6]. It is formed by well-normalized local histograms of image gradient orientations in a dense grid. The algorithm starts with dividing the image into $n$ small cells. For each cell, an $m$-bin HOG is built. Each bin in the HOG corresponds to an orientation spanning. The combination of $n$ histograms forms the HOG descriptor, with the size of $m \times n$ bins. To calculate the HOG, the intensity gradients are calculated first in both the horizontal and vertical directions. Then, the magnitude and the orientation of the gradient is calculated. Each gradient has a vote in its bin, which is its magnitude.

B. Model Training and Classification

The detector is built by feeding the sets of positive and negative samples into an SVM. For each detector and parameter combination, a preliminary detector is trained. Then the false positives are picked out and used as the “hard samples” to augment the training set. The model is then retrained using this augmented set to produce the final detector. As claimed in [6], this retraining process significantly improves the performance of the detector.

C. Implementation on the Embedded Smart Camera

Due to the limited resources of the embedded camera, the training stage is performed off-line on a PC. The size of each training sample is $64 \times 128$. A trained SVM is then imported onto the camera board.

For the HOG descriptor, unsigned orientations, ranging from 0 to 180 degrees, and nine histogram bins are used to achieve the best performance [6]. The size of the sliding window, the cells and the blocks are $64 \times 128$, $8 \times 8$ and $2 \times 2$, respectively. The overlapping between the blocks in the normalization step is 1. To handle the problem of different resolutions of the training samples and the test images, the test image is downsampled to multiple levels to search the possible positive detections. The final decision is made based on the distance between the HOG feature vector and the trained SVM vector. For the SVM, we use the OpenCV implementation.

During the detection phase, the binary classifier scans across the image at multiple scales and thus multiple overlapping detections are generated. These detections are fused by representing the detections in a position scale pyramid. A weighted point in the 3D space is generated by each detection and the weights represent the confidence of each detection. Then, a non-parametric density estimator is used to estimate the corresponding density function and the resulting modes of the density function form the final detections, with positions, scales and detection scores given by value of the modes.

The HOG calculation and classification are performed on the camera board. The frames are captured by the image sensor. Once a person is detected, a rectangle is drawn around it, and this frame is saved in the camera.
D. Performance and Explorations

As mentioned in Section II, the microprocessor of the embedded smart camera node is a fixed-point processor, i.e. there is no hardware support for floating point operations. However, both HOG and SVM require a large number of floating point operations.

We have performed the floating point operations by a software module, which increases the processing time. One of our future works is making the program more memory-efficient to decrease the number of memory accesses and power consumption.

To improve the processing speed of the current algorithm, we explored different scenarios by adding reasonable assumptions based on the applications. For example, when the camera is static, we may only need to watch a region of interest (ROI) in the view. Then, we do not need to search through the whole frame. If the ROI is much smaller than the whole frame, the processing time of each frame decreases significantly. Similarly, when the camera is mobile, we may crop parts of the image depending on the application. We performed several experiments to test the effects of adding these assumptions for different scenarios. Details and results are presented in Section IV.

IV. Experimental Results

In this section, we present the performance of the HOG-based human detector running on static and mobile embedded smart cameras. We show the outputs and report the processing times on four different scenarios.

One thing to note is that target detection and tracking from videos captured by mobile cameras is a challenging and computationally expensive task even for powerful computer platforms. Yet, it needs to be performed on the embedded platforms, as the next step of having operational mobile embedded smart cameras. The presented results are promising, and provide insight on the capabilities and limitations.

We employ the rectangular HOG descriptor presented in [6]. In the implementation, we use unsigned orientations, ranging from 0 to 180 degree, and nine histogram bins. The calculated HOG descriptors are then classified using an SVM, which determines whether a certain region is a person or not. For the SVM, we use the OpenCV implementation. The images captured by the camera are of size $320 \times 240$.

As described in Section II, the microprocessor on the camera mote is a fixed-point processor. In other words, it does not have hardware support for floating point operations. The HOG-based detection algorithm involves floating point operations, which we handle at software level. This combined with the limited processing power of the microprocessor increases the processing time.

In the first scenario, we performed human detection on the whole frames captured by the camera mote. The camera mote was held by a person who was moving around. Figure 2 shows the detection results. As can be seen, even though the background is complex and continuously changing, the detector can successfully detect the person(s) in the scene. In Fig. 2b, two people in the scene are close to each other, and the detector is able to correctly detect both of them. In addition, in Figures 2a through 2c, we can observe some changes in illumination. The HOG-based detector is robust to these changes, and shadow effects. It takes 37 seconds to perform detection on a frame of size $320 \times 240$. As mentioned above, having no hardware support for floating point operations contributes to this.

In order to see the effect of the size of area that is processed, we implemented the second scenario. In this experiment, the camera is static and observes the door to detect when someone passes through the door. Since the door occupies a relatively small portion of the image, we cropped a whole frame into a smaller image. Figure 3 shows the detection results for this scenario. The size of the cropped region is $80 \times 170$. It takes around 4.5 seconds to perform the detection on the microprocessor, which is much faster than processing the whole frame. Therefore, by carefully choosing the region of interest (ROI), it is possible to decrease the processing time. However, such an assumption will limit the application to static cameras since the ROI will change as the camera moves.

In the third scenario, the CITRIC camera is mounted on a
Fig. 2: The detection results when the camera was held by a person who was moving around. The detector can handle cases where two people are close to each other, and is robust to illumination changes.

Fig. 3: Detection results when the camera is static and only a region of interest, the door in this case, is processed for detecting people. The processing time is reduced significantly.

remote-controlled car, and the car is driven around in a room. Figure 1b shows the setup used in our experiments. In this case, the camera is close to the ground since it was directly mounted on the car, resulting in an oblique view of people. No assumption is made on how the car could move, i.e. the car can be freely driven around the room. Such a scenario is of interest for robotic applications where a camera-equipped robot or vehicle roam around a space to detect targets. Since camera can move around in any direction, we did not crop the frames. Thus, the frame size for processing is $320 \times 240$, and it takes around 37 seconds to process a frame. Figure 4 shows the detection results for this scenario. We can see that the detector performs well in this scenario despite the cluttered background. Also note that even though the image in Fig. 4f is blurred due to camera movement, we can still successfully detect the person.

For the forth scenario, we kept the camera mounted on the car but used a tripod to lift the camera, and thus have a better field of view. The car is driven along a hallway. To be able to perform cropping and see its effect, two assumptions are made: (a) the car moves along the hallway along almost a straight line, (b) a person does not get too close to the camera. Since the camera is moving, we do not crop the frame in the horizontal direction, i.e. keep the full width of the frame. Under the above assumptions, we can crop the frame vertically. We have cropped 60 rows from the top of each frame. Thus, the size of the cropped image is $320 \times 180$. It takes 20 seconds to process a frame. Since we are using the full width of the frame, the processing time is longer than the second scenario. Figure 5 shows the detection results for this scenario. As can be seen, the detector can handle the changes in both color and illumination.

As seen in Figures 2 through 5, the detection results are very good, and the presented results provide insights on the capabilities and limitations of these resource-limited platforms for mobile processing. The processing speed would increase on an embedded processor with built-in hardware support for floating point operations.

V. CONCLUSION

Towards the goal of performing object detection and tracking with mobile embedded smart cameras, we have implemented a HOG-based human detector on a CITRIC camera mote that combines a camera sensor with a microprocessor. HOG-based detectors allow us to detect foreground objects with moving cameras, and are much more robust towards illumination changes, shadows and image blur. Ability to detect objects with moving cameras has application in different areas including robotics, surveillance and smart driving systems. The example application we have chosen in this paper is
patrolling hallways in a building for detecting people. We have mounted embedded smart cameras on remote-controlled cars, and driven them around. We have also used them as handheld cameras. We have provided output images and reported processing times when using static and mobile cameras for different scenarios. Due to the nature of the HOG-based algorithm, the speed of the microprocessor, and not having hardware support for floating point operations, processing a $320 \times 240$ image takes around 37 seconds. Depending on the application, we can crop input images in different ways, and thus decrease the processing time significantly. The presented results are very promising, and provide insight on the capabilities and limitations of these embedded platforms.

REFERENCES


