Color Texture-Based Object Detection: An Application to License Plate Localization

Kwang In Kim[1], Keechul Jung[2], and Jin Hyung Kim[1]

[1]Artificial Intelligence Lab
CS Department, Korea Advanced Institute of Science and Technology
Taejon, 305-701, Korea
{kimki,jkim}@ai.kaist.ac.kr
[2]Pattern Recognition and Image Processing Lab
CS and Engineering Department, Michigan State University
East Lansing, MI 48824-1226, USA
jungke@msu.edu

Abstract. This paper presents a novel color texture-based method for object detection in images. To demonstrate our technique, a vehicle license plate (LP) localization system is developed. A support vector machine (SVM) is used to analyze the color textural properties of LPs. No external feature extraction module is used, rather the color values of the raw pixels that make up the color textural pattern are fed directly to the SVM, which works well even in high-dimensional spaces. Next, LP regions are identified by applying a continuously adaptive meanshift algorithm (CAMShift) to the results of the color texture analysis. The combination of CAMShift and SVMs produces not only robust and but also efficient LP detection as time-consuming color texture analyses for less relevant pixels are restricted, leaving only a small part of the input image to be analyzed.

1 Introduction

The detection of objects in a complex background is a well-studied, but yet unresolved problem. This paper presents a generic framework for object detection based on color texture. To demonstrate our technique, we have developed a Korean vehicle license plate (LP) localization system that locates the bounding boxes of LPs with arbitrary size and perspective under moderate amounts of changes in illumination. We stress that the underlying technique is fairly general, and accordingly can also be used for detecting objects in other problem domains, where object of interest may not be perfectly planar or rigid.

The following presents challenges in LP detection problem and a brief overview of previous related work along with the objective of the present study.

S.-W. Lee and A. Verri (Eds.): SVM 2002, LNCS 2388, pp. 293-309, 2002.

1.1 License Plate Detection Problem

Detecting the LP of a vehicle is of great interest because it is usually the first step of an automatic LP recognition system whose possible applications include traffic surveillance system, automated parking lot, etc.

Korean LP is a rectangular plate with two rows of white characters embossed on a green background.[1] The problem of automatic LP detection is challenging, as LPs can have significantly variable appearances in image:

Variations in shape due to the distortion of plates and differences in characters embossed on the plates (Fig. 1a)

Variations in color due to similar but definitely different surface reflectance properties, change in illumination conditions, haze, dust on the plate, blurring occurred during image acquisition, etc. (Fig. 1a).

Size variations and perspective deformation caused by a change of sensor (camera) placement with respect to vehicles (Fig. 1b).

(a)

(b)

Fig. 1. Examples of LP image with different zoom, perspective, and other various imaging conditions

1.2 Previous Work

A number of approaches have been proposed for developing LP detection systems. For example, color (gray level)-based methods utilize the fact that LPs (when disregarding the embossed characters) in images often show unique and homogenous color (gray level). They segmented an input image according to color (gray level) homogeneity and analyzed the color or shape of each segment. Kim, et al. [1] adopted

[1] In fact, there are two more color configurations for Korean LPs (deep blue characters on yellow background: business purpose vehicles, white characters on orange background: industrial vehicles). However we are interested in that of the most popular kind of vehicles (white characters on green background: personal vehicles).

genetic algorithms for color segmentation and searched for green rectangular regions as LPs. To make the system insensitive to noise and variation in illumination condition, they encouraged the consistency of the labeling between neighboring pixels during color segmentation. Lee, et al. [2] utilized a neural network (NN) to estimate surface color of LP from given samples of LP images. All the pixels in the image are filtered by NN and *greenness* for each pixel is calculated. Then the LP region is identified by verifying green rectangular region using structural features. Crucial to the success of color (or gray level)-based method is color (gray level) segmentation stage. However solutions currently available do not provide a high degree of accuracy in natural scene.

On the other hand, edge-based methods are based on observation that characters embossed on LPs contrast with their background (plate region) in gray level. Then, LPs are found by searching for regions with such a high contrast. Draghici [3] searched for regions with high edge magnitude and verified them by examining the presence of rectangular boundaries. In [4], Gao and Zhou computed gradient magnitude and their local variance in an image. Then, regions with a high edge magnitude and high edge variance are identified as LP regions. Although efficient and effective in simple images, edge-based methods can hardly be applied to a complex image, since in this case background region can also show high edge magnitude or variance.

With an assumption that LP region consists of dark characters on a light background, Cui and Huang [21] performed spatial thresholding on an input image based on a Markov random filed (MRF), and detected characters (LPs) according to the spatial edge variances. Similarly, Naito, et al. [6] performed adaptive thresholding on the image, segmented the resulting binary image using the priori knowledge of character size in LPs, and detected a string of characters based on the geometrical property (character arrangement) of LPs. While the reported results with clean images were promising, these methods may degrade when LPs are partially shaded or stained as shown in Fig. 1a, since in these cases, binarization may not correctly separate characters from the background.

Another type of approach stems from the well-known method of (color) texture analysis. In [5], Park, et al. presented a LP detection method based on color texture. They adopted a NN to analyze the color textural properties of horizontal and vertical cross-sections of LPs in an image and performed projection profile analysis on the classification result to generate LP bounding boxes. Brugge, et al. [11] utilized discrete time cellular NNs (DT-CNNs) for analyzing textural properties of LPs. They also attempted to combine a texture-based method with an edge-based method. Texture-based methods are known to perform well even with noisy or degraded LPs and rather insensitive to variations in illumination condition; however, they are most often time consuming, as texture classification is inherently computationally dense.

1.3 Overview of Present Work

Crucial to the success of color texture-based LP detection are the following: construction of (1) classifier that can discriminate between color textures associated with different classes (LP and non-LP), (2) LP bounding box generation module that

operates on the classification results obtained from (1). Accordingly, a color texture-based LP detection problem can be conventionally divided into two sub-problems: classification and bounding box generation. Recently, a large number of techniques for analyzing color texture have been proposed [7, 8]. In this paper, we focus our attention on a support vector machine (SVM)-based approach that was introduced in [9] for texture classification. SVMs are a natural choice because of their robustness even in lack of training examples. The previous success of SVMs in texture classification [9] and other related problems [10, 13] also provided further motivation to use SVMs as a classifier for identifying LP regions. In addition to robustness, SVMs have another important advantage. Since they work well even in high-dimensional spaces, no external feature extractor is required to reduce the dimensionality of the pattern, thereby eliminating the need for a time-consuming feature extraction stage. In fact, SVMs can efficiently extract features within their own architecture using kernel functions [9].

After the classification, a LP score image is generated where each pixel represents the possibility of the corresponding pixel in the input image being part of a plate region. Then, the LP bounding boxes within the LP score image are identified by applying the continuously adaptive meanshift algorithm (CAMShift), which has already demonstrated its effectiveness in a face detection application [12]. The combination of CAMShift and SVM produces robust and efficient LP detection through restricting the color texture classification of less relevant pixels. In addition to exhibiting a better detection performance than existing techniques, the processing time of the proposed method is short (average 1.4 second for 320×240-sized images).

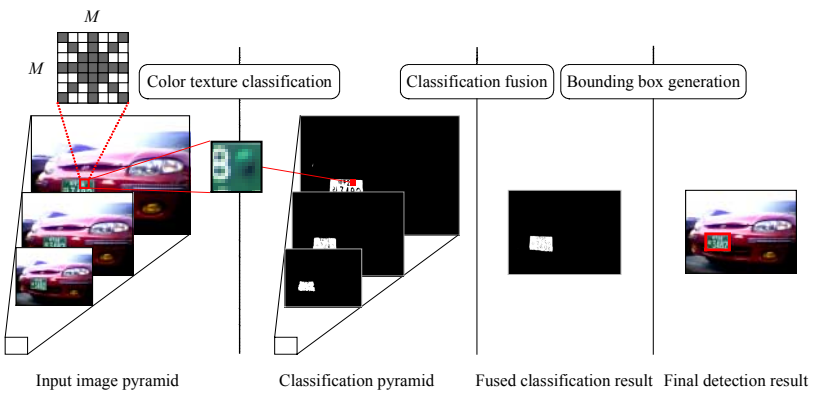


Fig. 2. Top-level process of LP detection system

2 System

The proposed method poses LP detection as a color texture classification problem where problem-specific knowledge is available prior to classification. Since this knowledge (the number and type of color textures) is often available in the form of example patterns, the classification can be supervised. A SVM as a trainable classifier

is adopted for this task. Specifically, the system uses a small window to scan an input image and classifies the pixel located at the center of the window into *plate* or *non-plate* (*background*) by analyzing its color and texture properties using a SVM.[2] To facilitate the detection of plate at different scales, a pyramid of images is generated from the original image by gradually changing the resolution at each level. The classification results are hypothesized at each level and then fused to the original scale. To reduce the processing time, CAMShift is adopted as a mechanism for automatically selecting the region of interest (ROI). Then, it locates bounding boxes of LPs by analyzing only these ROIs. Fig. 2 summarizes the LP detection process.

The rest of this section is organized as follows: Section 2.1 describes the use of SVMs for color texture classification. Next, Section 2.2 discusses the fusion of color texture analysis at different scales, while Section 2.3 outlines the LP detection process based on CAMShift.

2.1 Data Representation and Classification

One of the simplest ways to characterize the variability in a color texture pattern is by noting the color values (color coordinates in a color space e.g. RGB, HSI, etc.) of the raw pixels. This set of color values then becomes the feature set on which the classification is based. An important advantage of this approach is the speed with which the images can be processed since the features need not be calculated. Furthermore, this approach releases the system developer from the laborious feature design task. Yet, the main disadvantage is the large size of the feature vector. Accordingly, a classifier may be required to generalize the patterns in a high-dimensional space. In this case, it is important to keep the capacity of the classifiers as small as possible since classifiers with a larger capacity are inclined to store the specific details of a training set and can show a poor performance with a test set that differs in these details. SVMs provide a suitable means of constructing such classifiers, since they can simultaneously minimize the bound on the *capacity* of the classifier and the training error [13, 14]. Furthermore, SVMs incorporate feature extractors and can use their nonlinear mapped input patterns as features for classification [9]. Therefore, it may be more advantageous to allow a SVM to extract features directly from the pixels rather than forcing it to base the features on a user-defined feature set.

The architecture of a SVM color texture classifier involves three layers with entirely different roles. The input layer is made up of source nodes that connect the SVM to its environment. Its activation comes from the $M \times M$ (typically 11×11) window in the input image. However, instead of using all the pixels in the window, a configuration for autoregressive features (shaded pixels in Fig. 2) is used. This reduces the size of the feature vector (from $3 \times M^2$ to $3 \times (4M\text{-}3)$) and results in an improved generalization performance and classification speed. The hidden layer applies nonlinear mapping Φ from the input space to the feature space F and

[2] It is assumed that the number of texture classes is fixed at 2 for the purpose of object detection. While it is sometimes convenient to further divide the class of object of interest into more than two classes, when the intra-class variation (of object) is large, we assume that this two-class classification can be generalized to a multi-class classification.

computes the dot product between its input and *support vectors* (SVs) [13]. In practice, these two operations are performed in a single step by introducing the kernel function k which is defined as the dot product of two mapped patterns: ($k(\mathbf{x},\mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$). Accordingly, various mappings (feature extractions) Φ can be indirectly induced by selecting the proper kernels k. One feature extraction is achieved by taking the p-order correlations among the entries x_i of an input vector $\mathbf{x}$ [14]. If $\mathbf{x}$ represents a pattern of just pixel values, this amounts to mapping the input space into the space of p-th order products (monomials) of input pixels. It should be noted that the direct computation of this feature is not easy even for moderate-sized problems because of the extensive computation required. However, the introduction of a polynomial kernel facilitates work in this feature space, as the polynomial kernel with degree p ($k(\mathbf{x},\mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$) corresponds to the dot product of the feature vectors extracted by the monomial feature extractor C_p [14]:

$$(C_p(\mathbf{x}) \cdot C_p(\mathbf{y})) = \sum_{i_1,\ldots,i_p=1}^{N} x_{i_1} \cdot \ldots \cdot x_{i_p} \cdot y_{i_1} \cdot \ldots \cdot y_{i_p} = \left(\sum_{i=1}^{N} x_i \cdot y_i \right)^p = (\mathbf{x} \cdot \mathbf{y})^p .$$

The degree p is empirically determined to be 3. The size of the hidden layer m is determined as the number of SVs identified during in the training phase. The sign of the output y, obtained by weighting the activation of the hidden layer, then represents the class of the central pixel in the input window. For training, +1 was assigned for the LP class and –1 for the non-LP class. Accordingly, if the SVM output of a pixel is positive, it is classified as LP.

When using a learning-from-examples approach, it is desirable to make the training set as large as possible in order to attain a comprehensive sampling of the input space. However, when considering real-world limitations, the size has to be moderate. Accordingly, the problem is how to build a comprehensive yet tractable database. For plate patterns, a collection can be made of all the plate-containing images available. However collecting non-plate patterns is more difficult as practically any image can serve as a valid training example. From among these patterns a 'representative' set of non-plate patterns should selected. A *bootstrap* method recommended by Sung and Poggio [15] was adopted for this purpose. The idea is that some of the non-plate training patterns are collected during training rather than before training: a partially trained SVM is applied to images that do not contain LPs, then patterns with a positive output are added to the training set as non-plate patterns. This process iterates until no more patterns are added to the training set.

However, a training set constructed by a bootstrap, which is moderate (about 100,000 in preliminary experiments) for training conventional learning systems such as NNs, is often too large for training SVMs: when the number of training patterns is l, the training of a SVM requires an $O(l^2)$-sized memory, which grows prohibitively in proportion to l (when l>10,000 the memory grows to more than hundred gigabytes). Although several methods have been developed to reduce the time and memory complexity for training a large-scale SVM [16], they barely provide practical facilities for training sets larger than 100,000.

One straightforward method for training classifiers on such a large training set is *boosting by filtering*. As a type of boosting technique, it constructs a *strong* classifier based on a set of *weak* classifiers [17].[3] The method involves training several weak classifiers, which are different from each other in that they are trained on examples with different characteristics, and arbitrating the answers of the trained weak classifiers to make a strong decision. The training set for each classifier is iteratively obtained by utilizing the existing classifiers to *filter* a new training set. For a detailed description of boosting by filtering, readers are referred to [17]. A slight simplification is utilized in the current work: given a large training set, the first weak classifier is trained on a moderately sized subset of the training set. This classifier is then used to filter another set of examples to train the second classifier: those patterns for which the answers of the first classifier are wrong are collected. Next, the third training set is generated using the previous two filters (weak classifiers): those patterns for which the answers of both previous classifiers are wrong are collected. The procedure is iterated until no more patterns are collected. After the procedure is terminated, the answer for an unknown input pattern is obtained by voting among the classifiers. Fig. 3 summarizes the training process, while Fig. 4 shows examples of training images.

1. Create initial training set N_1 that includes complete set of *LP class* patterns N^L and partial set of non-plate patterns N^{NL}_1
   Train SVM E_1 on N_1
   I=2

2. **Do** {
   - Create training set N_i that includes N^L and random selection of 50% of patterns from N^{NL}_{i-1}
     Create empty set N^{NL}_i
   - **Do** {
     Exhaustively scan image which contains no LPs with $E_1,\ldots,E_{i-1}$ and add patterns whose corresponding $E_1,\ldots,E_{i-1}$ decisions are all *LP class* to N^{NL}_i
     } **Until** ($|N^{NL}_i| \geq 10{,}000$ or no more images are available)
   - Add N^{NL}_i to N_i
     Train SVM E_i on N_i
     i= i+1
   } **Until** (no more images are available)

Fig. 3. Classifier training process

[3] Degrees of 'Strong' and 'weak' are measured with respect to training errors.

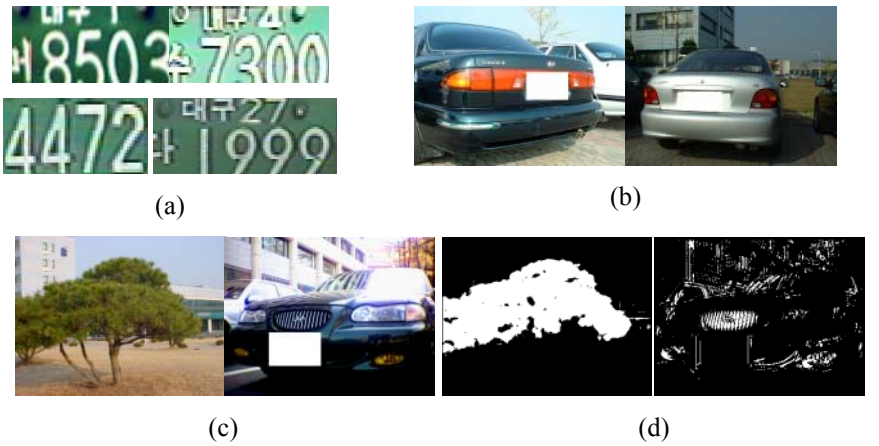


Fig. 4. Example images used in training: (a) LP images, (b) background images, (c) background images used in bootstrap, (d) classification results of images in (c) performed by E_1 (white: plate, black: background) (pixels showing green color or high-contrast were classified as LPs), where non-plate training patterns were tagged

2.2 Fusion of Classification at Different Scale

As outlined in Section 2.1, we do not use any external feature extractor for color texture analysis. Instead, the SVM receives color values of small input windows as a pattern for classification. While proven to be general and effective [9], this approach has an important shortcoming: it lacks of any explicit mechanism to fit variations in texture scale. This can be dealt with by constructing a sufficiently large set of training patterns sampled at various different scales. However, it is often more efficient to cast the problem into where the object of interest is size-invariantly represented.

A pyramidal approach is adopted for this purpose. In this method, the classifier is constructed at a single scale. Then in the classification stage, a pyramid of images is constructed by gradually changing the resolution of input image, and the classification is performed at each pyramid level. A mechanism is required to fuse the resulting pyramid of classification result (classification hypotheses at each level) into a single scale classification. We use a method similar to the NN-based arbitration which has already shown effectiveness in a face detection application [18].

The basic idea is to train a new classifier as an arbitrator to collect the outputs of classifiers at each level. For each location of interest (i, j) in the original image scale, the arbitrator examines corresponding locations in each level of output pyramid, together with their spatial neighborhood (i', j') ($|i'-i| \le N$ $|i'-i| \le N$, where N defines neighborhood relations and is empirically determined to be 3). A linear SVM is used as the arbitrator. Then, it receives the normalized outputs of color texture classifiers in a 3×3 window, around the location of interest at each scale. Normalization is simply done by applying a sigmoidal function to the output of SVM.

2.3 Plate Bounding Box Generation Using CAMShift

In most object detection applications, the detection process needs to be fast and efficient so that objects can be detected in real time while consuming as few system resources as possible. However, many (color) texture-based object detection methods suffer from the considerable computation involved. The majority of this computation lies in texture classification, therefore, if the number of calls for texture classification can be reduced, this will save computation time.[4] The proposed speed up approach achieves this based on following two observations:

(1) LPs (and often other different class of objects) form smooth boundary and (2) often, LP size does not dominate in the image.

The first observation indicates that LP regions may show aggregate of pixels showing LP specific characteristics (color texture) in the image. This validates the use of *coarse-to-fine* approach for local feature-based object detection: firstly, the ROI related to the possible object region is selected based on a coarse level of classification (sub-sampled classification of image pixels). Then only pixels in the ROI are classified at the finer level. This may significantly reduce the processing time when the object size does not dominate the image size (as supported by the second observation). It should be noted that the prerequisite for this approach is that the object of interest must be characterized with local features (e.g. color, texture, color texture, etc). Accordingly, features representing global characteristics of object, such as the contour or geometric moments may not be directly applied.

The implementation is borrowed from well-developed face detection methodologies. CAMShift was originally developed by Bradski [12] to detect and track faces in a video stream. As a modification of the meanshift algorithm that climbs the gradient of a probability distribution to find the dominant mode, CAMShift locates faces by seeking the modes of *flesh probability distribution*.[5] The distribution is defined as a two-dimensional image $\{y_{i,j}\}_{i,j=1,\ldots,IW,IH}$ (*IW*: image width, *IH*: image height) whose entry $y_{i,j}$ represents the probability of a pixel $x_{i,j}$ in the original image $\{x_{i,j}\}_{i,j=1,\ldots,IW,IH}$ being part of a face, and is obtained by matching $x_{i,j}$ with a facial color model. Then, from the initial search window, CAMShift iteratively change the location and size of window to fit its contents (or flesh probability distribution within the window) during the search process. More specifically, the center of window is moved to the mean of the local probability distribution and the size of window varies with respect to the sum of the flesh probabilities within the window, until no further considerable movement is observed. For the purpose of locating facial bounding box, the shape of search window is set as a rectangle [12]. Then, after iteration finishes, the finalized search window itself represents the

---

4 One simple method is *shifting* which classifies pixels at an interval and interpolates pixels located between them [20]. While significantly reducing the processing time, this technique trades between the precision and speed, and accordingly is not considered in this paper.

5 Actually, it is not a probability distribution, because its entries do not total 1. However, this is not generally a problem with the objective of peak (mode) detection.

bounding box of face in the image. For a more detailed description of CAMShift readers are referred to [12].

The proposed method is simply replacing the flesh probability $y_{i,j}$ with a *LP score* $z_{i,j}$ that is obtained by performing color texture analysis on the input $x_{i,j}$, and operating CAMShift on $\{z_{i,j}\}_{i,j=1,\ldots,IW,IH}$. Although the output of classifier (scale arbitrator: linear SVM) is not a probability (it is not even bounded), when it is scaled into an interval [0, 1] using a sigmoidal activation function, the detection results with CAMShift are acceptable. Henceforth, for convenience, these scaled classification results for the pixels within a selected window W will be referred to as the 'probability distribution within W'.

As a gradient ascent algorithm, CAMShift has possibility of being stuck to local optima. To resolve this, it is used in parallel with different initial window positions. This also facilitates the detection of multiple objects in an image.

One important advantage of using CAMShift on color texture-based object detection is that CAMShift does not necessarily require all the pixels in the input image to be classified. Since CAMShift utilizes local gradient, only the probability distribution (or classification result) within the window is sufficient for iteration. Furthermore, since the window size varies in proportional to the probabilities within the window, the search windows initially located outside of LP region may diminish, while windows located within the LP region grow. This is actually a mechanism for automatic selection of the ROI.

The parameters controlled in CAMShift at iteration t are the position $x(t)$, $y(t)$, width $w(t)$, height $h(t)$, and orientation $\theta(t)$ of the search window. $x(t)$ and $y(t)$ can be simply computed using moments:

$$x = M_{10} / M_{00} \text{ and } y = M_{01} / M_{00}, \tag{1}$$

where M_{ab} is the $(a+b)$-th moment as defined by

$$M_{ab}(W) = \sum_{i,j \in W} i^a j^b z_{i,j} .$$

$w(t)$ and $h(t)$ are estimated by considering the first two eigenvectors (two major axes) and their corresponding eigenvalues of the probability distribution within the window. These variables can be calculated using up to the second order moments [12]:

$$w = 2\sqrt{\left((a+c)+\sqrt{b^2+(a-c)^2}\right)/2}, \; h = 2\sqrt{\left((a+c)-\sqrt{b^2+(a-c)^2}\right)/2}, \tag{2}$$

where the intermediate variables a, b, and c are

$$a = M_{20} / M_{00} - x^2, \; b = 2\left(M_{11} / M_{00} - xy\right), \text{ and } c = M_{02} / M_{00} - y^2 .$$

Similarly, the orientation θ can also be estimated by considering the first eigenvector and corresponding eigenvalue of the probability distribution and then calculated using up to the second order moments as follows:

$$\theta = \arctan\left(\frac{b}{a-c}\right)/2 . \quad (3)$$

Note that the moments involve sums of all the pixels, and so are robust against small changes of elements. As such, a robust estimation of the parameters is possible even with the existence of noise (mis-classifications) in the window. Nonetheless, the use of equation (2) to estimate the window size is not directly suitable for the purpose of LP detection. When the whole LP region is completely contained within the search window, equ. (2) produces exactly what is needed. However, when the actual LP region is larger than the current (iteration t) window, the window size needs to be increased beyond the estimation of equ. (2) so as to explore a potentially larger object area in the next iteration (t+1). For this, h and w are set to be slightly larger than that estimated by equ. (2):

$$w = \alpha_w * 2\sqrt{\left((a+c)+\sqrt{b^2+(a-c)^2}\right)/2}\ ,\ h = \alpha_h * 2\sqrt{\left((a+c)-\sqrt{b^2+(a-c)^2}\right)/2}\ , \quad (4)$$

where α_w (≥1) and α_h (≥1) are constants determined to be 1.5. The new estimate enables the window to grow as long as the major content of the window is LP pixels. When the iteration terminates, the final window size is re-estimated using equ. (2).

The terminal condition for iteration is that for each parameter, the difference between two parameters of $x(t+1)$–$x(t)$, $y(t+1)$–$y(t)$, $w(t+1)$–$w(t)$, $h(t+1)$–$h(t)$, and $\theta(t+1)$–$\theta(t)$ in two consecutive iterations (t+1) and (t) is less than the predefined thresholds T_x, T_y, T_w, T_h, and T_θ respectively.

During the CAMShift iteration, search windows can overlap each other. In this case, they are examined as to whether they are originally a single object or multiple objects. This is performed by checking the degree of overlap between two windows, which is measured using the size of the overlap divided by the size of each window.

Supposing that D_α and D_β are the areas covered by two windows α and β, then the degree of overlap between α and β is defined as

$$\Lambda(\alpha,\beta) = \max\left(\text{size}(D_\alpha \cap D_\beta)/\text{size}(D_\alpha), \text{size}(D_\alpha \cap D_\beta)/\text{size}(D_\beta)\right),$$

where $\text{size}(\lambda)$ counts the number of pixels within λ. Then, α and β are determined to be

a single object if $T_o \le \Lambda(\alpha,\beta)$

multiple objects otherwise,

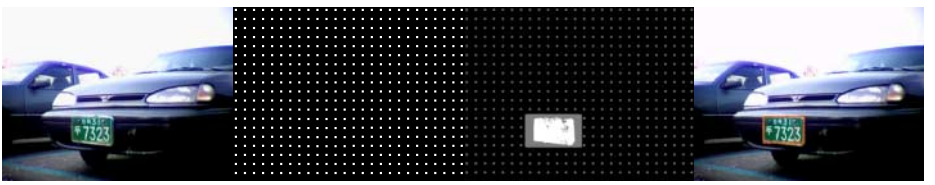
where T_o is the threshold set at 0.5.

Then, in CAMShift iteration, every pair of overlapping windows is checked and those pairs identified as a single object are merged to form a single large window encompassing them. After CAMShift iteration finishes, any small windows are eliminated, as they are usually false detections.

1. Set up initial locations and sizes of search windows *W*s in image.
   For each *W*, repeat Steps 2 to 4 until terminal condition is satisfied.
2. Generate LP probability distribution within *W* using SVM.
3. Estimate parameters (location, size, and orientation) of *W* using eqns. (1), (3), and (4).
4. Modify *W* according to estimated parameters.
5. Re-estimate sizes of *W*s using equ. (2).
6. Output bounding *W*s as LP bounding boxes.

Fig. 5. CAMShift for LP detection

Fig. 5 summarizes the operation of CAMShift for LP detection. It should be noted that, in case of overlapping windows, the classification results are cached so that the classification for a particular pixel is performed only once for an entire image. Fig. 6 shows an example of LP detection. A considerable number of pixels (91.3% of all the pixels in the image) are excluded from the color texture analysis in the given image.



(a) (b) (c) (d)

Fig. 6. Example of LP detection using CAMShift: (a) input image (640×480), (b) initial window configuration for CAMShift iteration (5×5-sized windows located at a regular interval of (25,25)), (c) color texture classified region marked as white and gray levels (white: LP region, gray: background region), and (d) LP detection result

3 Experimental Results

The proposed method was tested using a LP image database of 450 images, of which 200 were stationary vehicle images taken at the parking lots and the remaining 150 were taken from the vehicles traveling on a road. The images included examples of LP appearances varying in terms of size, orientation, perspective, illumination condition, etc. The resolution of these images ranged from 240×320 to 1024×1024 while the sizes of LPs in these images range from about 79×38 to 390×185. One hundred randomly selected images from the database were used for collecting the initial samples for training the SVMs, and the remaining 350 images were used for testing. Non-LP training examples for bootstrapping the SVMs were also collected from 75 images that contained no LPs and were distinct from the LP image database.

For each detected LP region, the system drew a rectangle encompassing that region in the input image.

The initial locations and sizes of the search windows are dependent on the application. A good selection of initial search windows should be relatively dense and large enough not to miss LPs located between the windows and to be tolerant of noise (classification errors), yet they also should be moderately sparse and small enough to ensure fast processing. The current study found that 5×5-sized windows located at a regular interval of (25, 25) were sufficient to detect LPs.

Variations in the parameter values of CAMShift (for termination condition) did not significantly affect the detection results excepting the case that they were so large as to make the search process converge prematurely. Therefore, based on various experiments with the training images, the threshold values were determined as T_x=T_y=3 pixels, T_w=T_h=2 pixels and T_θ=1°. The slant angle θ of the finalized search windows was set at 0° if its absolute value was less than 3°, and 90° if it was greater than 87° and less than 93°. This meant that small errors occurring in the orientation estimation process would not significantly affect the detection of horizontally and vertically oriented LPs. Although these parameters were not carefully tuned, the results were acceptable with both the training and test images, as shown later.

The systems were coded in C++ and run on a Pentium3 CPU with an 800 MHz clock speed. The time spent processing an image depended on the image size and number and size of LPs in the image. Most of the time was spent in the classification stage. For the 340×240-sized images, an average of 12.4 seconds was taken to classify all the pixels in the image. However, when the classification was restricted to just the pixels located within the involved search windows of CAMShift, the entire detection process only took an average of 1.4 second.

To quantitatively evaluate the performance, an evaluation criteria need to be established for the detections produced automatically by the system. A set of ground truths *GT*s is created by manually constructing bounding boxes around each LP in the image. Then, the outputs produced by the system *A*s are compared with the *GT*s. This comparison is made by checking the similarity, defined as the size of the overlap divided by the size of the union. Formally, the similarity between two detections α and β is defined as

$$\Gamma(\alpha,\beta)=\text{size}\left(D_\alpha \cap D_\beta\right)/\text{size}\left(D_\alpha \cup D_\beta\right).$$

Actually, this is the Tanimoto similarity [19] between two detections when they are represented as *IW*×*IH* vectors whose elements are 1 if the corresponding pixels are contained within the detection areas and 0, otherwise. As such, the output of the comparison is a binary value of $\{correct, incorrect\}$, which is determined as

correct, if $T<\Gamma(GT, A)$,

incorrect, otherwise,

where *T* is the threshold, which is set at 0.8.

The incorrect detections are further classified into {*false detection*, *miss*} as defined by:

false detection, if $\text{size}(D_A) - \text{size}(D_{GT}) > 0$,

miss, otherwise.

Two metrics are used to summarize the detection results, as defined by:

$$\text{miss rate}(\%) = \frac{\#\text{misses}}{\#\text{LPs}} \times 100,$$

$$\text{false detection rate}(\%) = \frac{\#\text{false detections}}{\#\text{LPs}} \times 100.$$

The proposed system achieved a miss rate of 2.6% with a false detection rate of 9.4%. Almost all the plates that the system missed were blurred during imaging process, stained with dust, or strongly reflecting the suhshine. In addition, many of the false detections were image patches with green and white textures that looked like parts of LPs. Figs. 7 and 8 show examples of LP detection without and with mistakes, respectively.

Fig. 7. LP detection examples: The system exhibited a certain degree of tolerance with pose variations ((c), (h), and (l)), variations in illumination condition ((a) compared with (d) and (e)), and blurring ((j) and (k)). Although (d) shows fairly strong reflection of sunshine and accordingly show quite different color properties (or irradiance) from surface reflectance, our LP detector correctly located it. While green leaves are originally classified into LPs before bootstrap (Fig. 4(c)), they were correctly classified into background and were not detected as LP ((a) and (l))

To gain a better understanding of the relevance of the results obtained using the proposed method, benchmark comparisons with other methods were carried out. A set of experiments was performed using a color-based method [2] and color texture-based method [5], which are described in Section 1.2. Since both methods are developed to detect horizontally aligned LPs without perspective, the comparison were made with 82 images containing upright frontal LPs.

Table 1 summarizes the performances of the different systems: A (SVM+CAMShift) was the proposed method, B (SVM+profile analysis) used a SVM for color texture classification and profile analysis [5] for bounding box generation, C (NN+CAMShift) used a NN adopted from [5] for classification and CAMShift for bounding box generation, and D (NN+profile analysis) and E (Color-based) were the methods described in [5] and [2], respectively.

A and B exhibited similar performances and were far better than the other three methods, while A was much faster than B. C and D also showed similar performances and C was faster than D. They were rather sensitive to changes in illumination condition than A and B. While showing the highest processing speed, E produced the highest miss rate, which mainly stemmed from a poor detection of LPs reflecting sun light or with strong illumination shadow that often occurred in natural scene.

(a) (b) (c) (d)

Fig. 8. Examples of LP detection with mistakes: LPs were missed due to bad illumination (a) and excessive burring (b). False detections are present in (c) and (d) where white characters were written on green background and complex color patterns occurred in glass, respectively

Table 1. Performances of various systems

System	Miss rate	False detection rate	Avg. proc. time per image
A	3.7(%)	7.3(%)	1.28(sec.)
B	3.7(%)	9.8(%)	9.03(sec.)
C	8.5(%)	13.4(%)	0.69(sec.)
D	7.3(%)	17.1(%)	3.92(sec.)
E	20.7(%)	26.8(%)	0.35(sec.)

4 Conclusions

A color texture-based method for detecting LPs in images was presented. The system analyzes the color and textural properties of LPs in images using a SVM and locates their bounding boxes using CAMShift. While the proposed method does not assume the size and perspective of LPs, and is relatively insensitive to variations in illumination, it also can facilitate fast LP detection and was found to produce a better performance than some other techniques. However, the proposed method encountered problems when the image is extremely blurred or quite complex in color.

While many objects can effectively be located with bounding box, there are some objects whose location cannot be fully described by only bounding box. When the precise boundaries of these objects are required, more delicate boundary location method can be utilized. Possible candidates include deformable template model [7]. Then the object detection problem can be dealt with fast and effective ROI selection process (SVM+CAMShift) followed by delicate boundary location process (such as deformable template model).

5 References

1. Kim, H. J., Kim, D. W., Kim, S. K., Lee, J. K.: Automatic Recognition of A Car License Plate using Color Image Processing. Engineering Design and Automation Journal 3 (1997)
2. Lee, E. R., Kim, P. K., Kim, H. J.: Automatic Recognition of A License Plate Using Color. In Proc. International Conference on Image Processing (ICIP) (1994) 301-305
3. Draghici, S.: A Neural Network based Artificial Vision System for License Plate Recognition. International Journal of Neural Systems 8 (1997) 113-126
4. Gao, D.-S., Zhou, J.: Car License Plates Detection from Complex Scene. In Proc. International Conference on Signal Processing 2000 1409-1414
5. Park, S. H., Kim, K. I., Jung, K., Kim, H. J.: Locating Car License Plates using Neural Networks. IEE Electronics Letters 35 (1999) 1475-1477
6. Naito, T., Tsukada, T., Yamada, K., Kozuka, K., Yamamoto, S.: Robust License-Plate Recognition Method for Passing Vehicles under Outside Environment. IEEE Trans. Vehicular Technology 49 (2000) 2309-2319
7. Zhong, Y., Jain, A. K.: Object Localization Using Color, Texture and Shape. PR. 33 (2000) 671-684
8. Mirmehdi, M., Petrou, M.: Segmentation of Color Textures. IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI) (2000) 142-159
9. Kim, K. I., Jung, K., Park, S. H., Kim, H. J.: Support Vector Machines for Texture Classification. IEEE Trans. PAMI to be appeared
10. Osuna, E., Freund, R., Girosi, F.: Training Support Vector Machines: an Application to Face Detection. In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (1997) 130-136

11. Brugge, M. H. ter, Stevens, J. H., Nijhuis, J. A. G., Spaanenburg, L.: License Plate Recognition using DTCNNs. In Proc. IEEE International Workshop on Cellular Neural Networks and their Applications (1998) 212-217
12. Bradski, G. R.: Real Time Face and Object Tracking as a Component of a Perceptual User Interface. In Proc. IEEE Workshop on Applications of Computer Vision (1998) 214-219
13. Vapnik, V.: The Nature of Statistical Learning Theory. Springer-Verlag, NY (1995)
14. Schölkopf, B., Sung, K., Burges, C. J. C., Girosi, F., Niyogi, P., Poggio, T., Vapnik, V.: Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. IEEE Trans. Signal Processing 45 (1997) 2758-2765
15. Sung, K. K., Poggio, T.: Example-based Learning for View-based Human Face Detection. IEEE Trans. PAMI 20 (1998) 39-51
16. Joachims, T.: Making Large-Scale SVM Learning Practical. In Schölkopf, B., Burges, C., Smola, A. (Eds.): Advances in Kernel Methods-Support Vector Learning, MIT-Press, Cambridge (1999) pp. 169-184
17. Haykin, S.: Neural Network-A Comprehensive Foundation, 2nd edn. Prentice Hall, NJ (1999)
18. Rowley, H. A., Baluja, S., Kanade, T.: Neural Network-based Face Detection. IEEE Trans. PAMI. 20 (1999) 23-37
19. Duda, R. O., Hart, P. E.: Pattern Classification and Scene Analysis, A Wiley-interscience publication, NY (1973)
20. Li, H., Doermann, D., Kia, O.: Automatic Text Detection and Tracking in Digital Video. IEEE Trans. Image Processing 9 (2000) 147-156
21. Cui, Y., Huang, Q.: Extracting Characters of License Plates from Video Sequences. Machine Vision and Applications 10 (1998) 308-320