Universidad Autónoma de Madrid
Departamento de Ingeniería Informática
Escuela Politécnica Superior

VOTING METHODS

FOR

GENERIC OBJECT DETECTION

TRABAJO FIN DE MÁSTER
MAY 2013

AUTOR: QIJIONG JIANG
DIRECTOR: GONZALO MARTÍNEZ MUÑOZ

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

In the last years, image recognition has been a booming field of research because of the huge demand of derivative applications. However, the detection and recognition of objects in images are difficult problems. One of the main problems is that the localization of the object in the images could be in scenes with high level of variegation and clutter, where the objects of interest are similar to the background, objects are partially visible or the sizes of the objects are very small with respect to the size of the images. In this context, two important subareas of image recognition are: image classification and object detection.

In recent years many methods have been proposed for image classification that have improved the performance drastically. Among them are the methods for image classification based on the creation of supervised visual dictionaries [Nowak *et al.* (2006); Willamowski *et al.* (2004); Torralba *et al.* (2007)]. The first algorithm of this type applied to image classification is [Csurka *et al.* (2004)]. In [Csurka *et al.* (2004)], the authors presented the Bag of Words (BOW) method to classify images. This method treats each image as a document and each image feature as a word. Visual words could be considered as a representative of several similar image visual feature patches. Image visual features are transformed to their most similar visual word. For each image, a vector of occurrence counts of words is created. This vector is used to compute the similarity between images. In [Winn *et al.* (2005)], the authors propose using a high number of clusters using K-means, these clusters then are concatenated to maximize the generalization power of the model. In [Moosmann *et al.* (2007)], the authors build extremely randomized clustering forests which classify each image directly using the output as the descriptor vector. After that, the leaves of the trees are defined as the binary attributes which are used as the entries of the visual dictionary. This method is discriminative, however, the creation of a dictionary with large number of entries might overfit the data. In [Sukthankar2 (2008)], it is proposed to use a sequence of ”visual bits” using Boosting algorithm. Each visual bit is a binary attribute that is the entry for the next classifier in the ensemble. In a recent work [Martínez-Muñoz *et al.* (2009)], a free-dictionary method is proposed to classify images. This method analyzes directly the descriptor vectors of the image using Random Forests. Instead of using the trees as the entries of a visual dictionary, they are used as a data structure. Each tree stores a histogram of the number of reached training examples of each class in each leaf. In test, the histograms are accumulated for all feature descriptors of a new image. The global accumulated histogram is passed into a new classifier that outputs the final prediction. This approach avoids the loss of information that occurs when each feature descriptor is mapped to a visual dictionary entry. This is a non-parametric method and it is called evidence trees reflecting the base mechanism.

Another important area of research in computer vision area is object detection. There are multiple object detection techniques. Some of them are based on sliding window detectors [Viola and Jones (2004)], contour dictionaries [Shotton *et al.* (2005)], modeling of objects with constellations of parts using local image parts (patches) [Fergus *et al.* (2003)] or contour fragments of objects [Opelt *et al.* (2006)]. The Generalized Hough Transformation for the shape detection is also used with the combination of dictionaries of local attributes [Leibe *et al.* (2008)]. In [Maji and Malik (2009)], the authors propose a discriminative version of probabilistic Hough transformation. In [Gall and Lempitsky (2009)], it is proposed to use Hough forests, where each tree votes the region of the image where a possible object centroid might be localized. This method also uses a generalized version of the Hough transformation with discriminative learning. The focus of this project is the adaptation of image classification methods for object detection. Specifically, it aims to bring the system of evidence trees for locating the centroids of the objects in a object detection problem.

Ensemble learning is an area that there has been a lot of research [Breiman (1996); Freund and Schapire (1996); Hansen and Salamon (1990); Martínez-Muñoz and Suárez (2005)]. In ensemble methods, a collection of classifiers are created and their decisions are combined to obtain the final prediction. Instead of considering a single prediction hypothesis, these methods take into account the predictions of various hypotheses which are consistent according to the observed data. Averaging these predictions provides a mechanism to obtain more accurate and robust final decisions. There is ample empirical evidence that the generated final predictions have good generalization capabilities. In recent years, these powerful machine learning techniques have been applied in problems of detection and classification in images [Moosmann *et al.* (2007); Gall and Lempitsky (2009); Martínez-Muñoz *et al.* (2009)].

We believe that some ideas of image classification algorithms can be applied in an object detection algorithm. Tree evidences approach proposed in [Martínez-Muñoz *et al.* (2009)] was used for a classification algorithm. In this work, we propose to adapt this idea to object detection. The idea of vote maps used in detection algorithms [Opelt *et al.* (2006); Shotton *et al.* (2005); Gall and Lempitsky (2009)] is applied to detect the center of an object. The mentioned algorithms used the geometric information like the relative distance between the image feature and the location of the object of interest. This approach of using geometric information is also an interesting research area for this work. On the other hand, we will use ensemble learning algorithm such as Random Forests to combine location predictions for an object and use the final result to annotate the center of an object.

This research has been carried out in collaboration with professor Thomas G. Dietterich of Oregon State University and professor Qing Yao of Zhejiang Sci-Tech University.

1.2 Objectives

The Master's Thesis has following objectives:

- To study the state of the art in image classification and object detection algorithms.
- To study the incorporation of machine learning techniques such as ensembles of classifiers for object detection problems in images.
- To analyze the performance of object detection methods based on visual dictionaries in object detection and to design new algorithms that incorporate randomization techniques and supervised combination of a set of classifiers to generate visual dictionaries.
- To incorporate geometric constraints in object detection and to design a new generic object detection algorithm that is useful for any kind of objects.
- To evaluate and compare the proposed algorithm with previous state of the art detection algorithm.

The main objective is the design of an algorithm for combining multiple object location detection decisions obtained by using visual dictionaries and machine learning algorithms in a single final decision. In the same time, the design and implementation of the testing protocol have to be done to evaluate the algorithm performance on different datasets. A possible application for biomonitoring of ecosystems is investigated also. For example, using the proposed algorithm to detect insect locations in an image or to identify a group of high similarity species.

1.3 Document Structure

The structure of the Thesis is as follows:

- Chapter 1. This chapter presents the motivation and the objectives of this work.
- Chapter 2. This chapter presents some state of the art algorithms which inspired us in our own works.
- Chapter 3. This chapter presents the design and the details of the proposed object detection method.
- Chapter 4. This chapter presents experimental results of the proposed algorithm in different dataset and the comparison results with other methods.
- Chapter 5. This chapter summarizes the main results and conclusion of the work, some related future works will be suggested.

Chapter 2

Related works

The main objective of an image recognition system is to transform the information contained in an image to some computer format that can be utilized for some automatic processing. Some of the typical processing in this area could be: image classification, object detection, image retrieval, medical image analysis, etc. And the output of these systems can be some discrete values indicating the class which an image belongs to (classification) or the position of an detecting object with the area in which the object occupies in the image (detection).

In computer vision area, feature extraction is always a crucial step. In this process the physic of the image is transformed to some low level signals which are interpreted by a recognition system. Therefore extraction of irrelevant or noisy features leads to the failure of entire object recognition system. Good image features are those invariant to changes in: lighting conditions, rotation, scaling, affine transformations and partial occlusion. On the other hand, for classification system, a good features have to preserve high discrimination between different images or objects. The metric which measures the effectiveness of an image feature is called ”repeatability” [Lowe (2004)]. A good image feature has to have high repeatability which means: Similar features will have to be detected in different images of the same scene or object.

Image features are in a sense like human eyes, we use them to perceive the environment and situation around us. We used images features to transform the external images in some computer signals or statistics, this is what we do with our eyes to transform external environment to some visual stimulations and transfer them to our brain. Without good image features we are like blind people trying to read. However, we also need a machine brain to understand the perceived information. For this purpose, a lot of machine learning algorithms have been applied in this area to make our agent smart enough for a specific image recognition tasks.

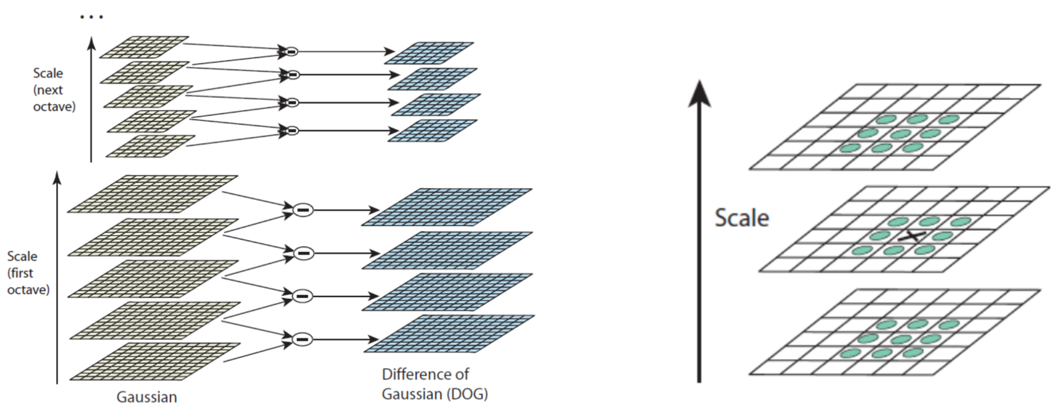


Figure 2.1: SIFT Detector. Left, images are scaled in different sizes. In each octave, images are filtered in different smoothness and DoGs are computed in each octave. Right, the keypoint (cross) is detected as the maximum of its neighbours (circles) in the DoG.

In this section we give a revision of some of the most used algorithms and techniques for image recognition tasks. Significant emphasis is given to some of them which will be implemented and used in our approach or used as a basis for performance comparisons.

2.1 Feature detection

2.1.1 SIFT feature

In 1999 Lowe proposed a image feature named feature transform (SIFT) [Lowe (1999)]. Unlike other previous global image features extraction that were based on the characteristics of the whole image like global texture and colour features, SIFT feature is a patch-based feature which is used to detect a set of singular circular regions in an image. Each individual SIFT feature presents robustness to partial occlusion, scale, illumination and orientation changes and is partially invariant to affine transformations. SIFT feature has also been demonstrated to be robust in detecting 3D objects from a large variation of viewpoints [Lowe (2001)]. These stabilities across different image conditions make the SIFT detector and descriptor suitable for image recognition problems. The SIFT algorithm includes three modules: a detector to detect invariant patches in the image, a descriptor to define the patches and a matching algorithm between SIFT descriptors.

The SIFT detection process works as following. First, the image is down-sampled by a factor of two a few times. The number of subsamplings determine the number of possible scales of the detected point. These scaled images are convolved using Gaussian filters with different variances (σ). The Gaussian filters are applied in order to smooth the image and to remove noisy pixels. The set of filtered images obtained from the same scale is called octave. In each octave, the difference of two adjacent layers is computed. This difference is called Difference of Gaussians (DoG).

The location of Keypoints are identified by finding the maxima and minima in the DoG images with respect to its neighbouring points. The neighbouring points are eight point around the maximum/minimum in its layer and the nine neighbouring points in its upper and lower layer. This process is shown in Figure (2.1).

Once the center position of each keypoint is detected, we have to obtain the scale and orientation of the patch around the keypoint. The size of the keypoint is determined by the scale of the octave in which it has been detected. The bigger the scale used, the smaller is the region occupied by the keypoint in the original image. The size of the keypoint region is a Gaussian window which is set to 1.5 times bigger than the Gaussian filter used in the smoothed image where the keypoint center is detected. The orientation of the keypoint is obtained by selecting the dominant gradient direction around the center of the keypoint. A histogram of gradient directions with 36 bins is created, each bin covers 10 degrees. The gradient direction θ of each sample position is weighted by the gradient magnitude m and a Gaussian-weighted circular window with a $sigma$ 1.5 times bigger than the scale of the detected keypoint:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2, (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = atan2(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y))$$

where operator $L(x,y)$ indicates the pixel value in the position x,y of the Gaussian-blurred image L. The dominant gradient directions are considered as those above 80% of the highest histogram maximum. Sometimes more than one dominant orientation is formed and more than one SIFT detection is assigned to the same position of the image.

The final step is to obtain a descriptor vector of the detected patch. The SIFT descriptor is needed to define, in a compact way, the region that the keypoint occupies. A SIFT descriptor is a histogram of a 128-dimensional vector and the histogram is an estimation of occurrences of gradient directions rounded of each inner sample point in discrete intervals (bins).

To construct it, a 16-by-16 rectangle is cropped by circular Gaussian-weighted window. The window used to compute the dominant orientation is used here. A keypoint descriptor is described by computing the gradient magnitude and orientation at each image sample

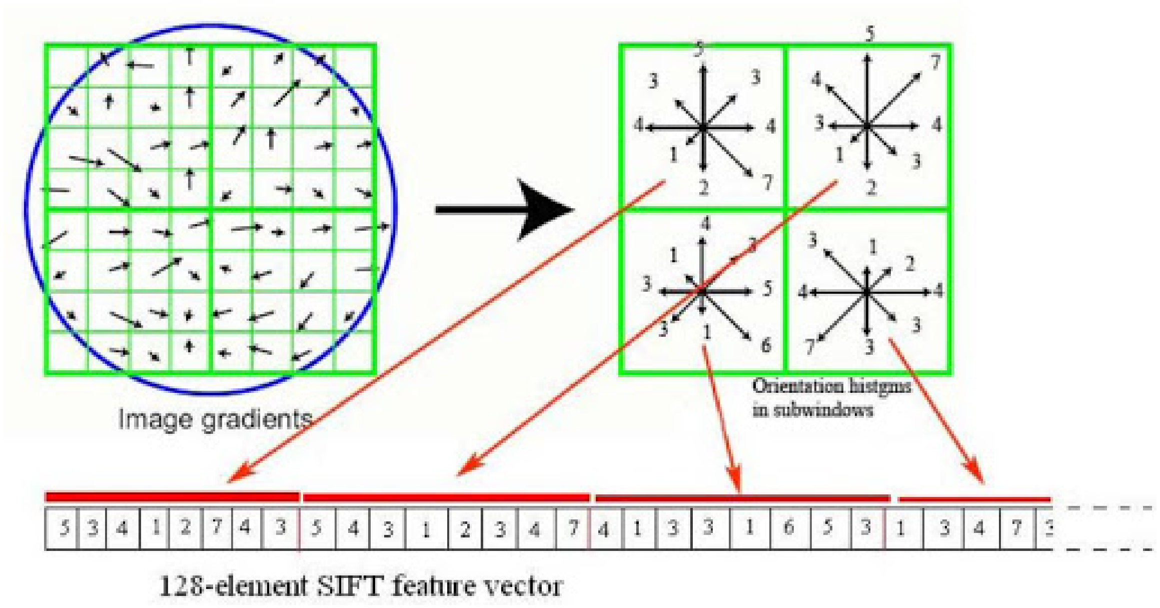


Image gradients
Orientation histgms in subwindows
128-element SIFT feature vector

Figure 2.2: SIFT descriptor.

point and the gradient magnitude is normalized by the Gaussian window. The orientation of each sample point is rounded into eight discrete directions. Note that the region around the keypoint position is normalized previously in orientation, i.e. rotated according to the orientation of the keypoint.

The patch region is divided in 16, 4-by-4 subregions. Each one generates a sub-histogram with eight that bins represent discrete orientations. This is done by accumulating the normalized gradient magnitude of each sample point in its respective orientation bin. By concatenating all these sub-histograms, we finally get a 128-dimensional descriptor vector. This process is shown in Figure (2.2).

The final module of the SIFT algorithm is the matching step of SIFT descriptors. This is done by identifying its nearest neighbour in the database of keypoints from training images. However, a typical image will produce several thousand overlapping features at different scales. Then for a database with a set of images, this step may be a bottleneck. In the original work [Lowe (2001)], the authors proposed to use a probabilistic version of the KD-tree algorithm [Beis and Lowe (1997)] to compute the descriptor matching efficiently.

2.1.2 Harris and Hessian affine detectors

Other keypoint detection methods are used and combined with SIFT descriptor for different object detection and classification problem. In [Mikolajczyk *et al.* (2005)], Harris affine detector and Hessian affine detector are presented. These detectors manage to find corners that present high affine invariance. The idea behind the algorithms of these detectors is to assume that corner regions in an image have large intensity changes in multiple directions.

The Harris affine detector is divided in two parts:

- Detect the feature region using scale-invariant Harris-Laplace Detector.
- Normalize the affine region and re-estimate the affine region by selecting keypoint locations and scale properly.

The Harris-Laplace Detector is a detector which applies first the Harris corner detector in multiple image scales. Using Harris corner detector in multiples scales makes the Harris-Laplace Detector scale invariant. Harris corner detector uses the second moment matrix A to determine corners in images as shown below:

$$A(X) = \sum_{x,y} w(x,y) \begin{vmatrix} I_x^2(X) & I_xI_y(X) \\ I_xI_y(X) & I_y^2(X) \end{vmatrix}$$

where $I_x(X)$ and $I_y(X)$ are derivatives of pixels intensity in the x and y direction at point X, I^2 is the second derivative, and $w(x,y)$ is a circular Gaussian which is a weighting function for each position (x,y). According to [Harris and Stephens (1988)], in corner regions, matrix A will have two large, positive eigenvalues λ_1 and λ_2. A measure of cornerness, R, can be computed as:

$$R = det(A) - \alpha \cdot trace^2(A) = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

where α is a constant. If R(x,y) is bigger than a specific threshold t, the position is considered a corner.

Up to this point the detector is not scale invariant. In order to make it scale invariant, it is necessary to apply this detection in the Gaussian at different scale spaces, similar to the ones used for SIFT. This combination is named Harris-Laplace detector. Taking the scale into account the measure R can be redefined as:

$$R = det(\mu(X, \sigma_I, \sigma_D)) - \alpha\, trace^2(\mu(X, \sigma_I, \sigma_D))$$

where $\mu(X, \sigma_I, \sigma_D)$ is:

$$\mu(X, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \bigotimes \begin{vmatrix} I_x^2(X, \sigma_D) & I_x I_y(X, \sigma_D) \\ I_x I_y(X, \sigma_D) & I_y^2(X, \sigma_D) \end{vmatrix}$$

and $\sigma_D^2 g(\sigma_I) \bigotimes$ denotes the convolution of the Gaussian kernel applied to each derivative and σ_I indicates the scale at which the Harris corner points are detected, i.e. the size of the Harris corner. And σ_D establishes the smoothness of the Gaussian kernel operator. Corner points are those local (eight point neighbourhood) maxima of the cornerness that are above a specified threshold. Then an automatic iterative scale selection process is done [Lindeberg (1998)]. The iterative algorithm both spatially localizes the corner points and selects the characteristic scale. This process has three key steps:

- Choose the scale σ_I which maximizes the Laplacian-of-Gaussians (LoG) [Wikipedia (2004c)] from a point X.
- Using the selected scale, the new point X is chosen such that it maximizes the Harris corner measure.
- Stop if σ_I no suffers changes between two consecutive iterations.

At this point, the detected multi-scale Harris corners are not affine invariant. However, these corners provide an approximate location and scale for initialization for the next iterative process. The next step is done to estimate the regions with affine properties. Finally, the affine regions are normalized with stretch and skew removed. Figure (2.3) shows an example of this iterative process.

The Hessian affine detector algorithm proposed by the same author is very similar to the Harris affine detector. The main difference between these two algorithms is that the Hessian affine detector detects interest points based on the Hessian matrix:

$$H(X) = \begin{vmatrix} L_{xx}(X) & L_{xy}(X) \\ L_{xy}(X) & L_{yy}(X) \end{vmatrix}$$

where the L_{aa} is the partial derivative in the direction a. Note that $L(x) = g(\sigma_I) \bigotimes I(x)$ and it means that the derivatives are smoothed by a Gaussian kernel. The scale of the detected interest point is determined by the σ_I. Once the initial Hessian points are detected, the same iterative affine region estimation-normalization process used in Harris-affine detector is done here to get Hessian affine regions.

Both detected Harris affine region and Hessian affine region can be described using the SIFT descriptor.

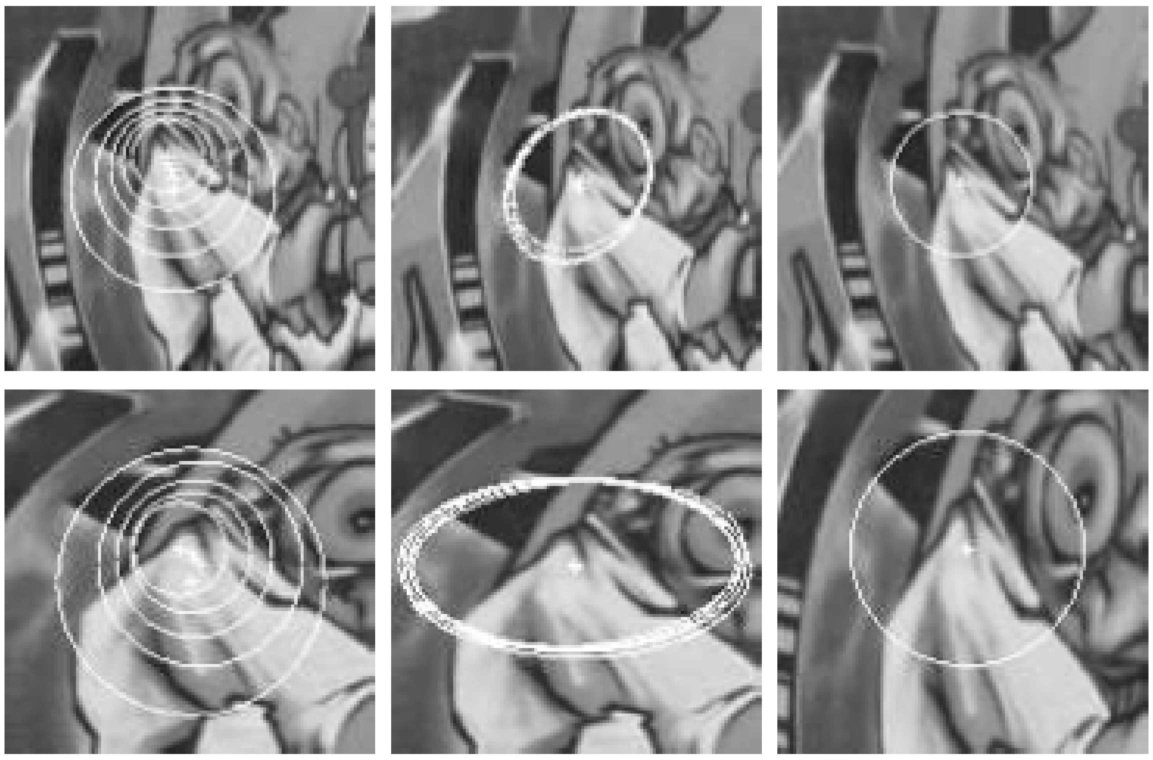


Figure 2.3: Left, the multi-scale Harris corner points detected by Harris-Laplace detector. Center, the affine region is estimated, Right, The affine region is normalized.

2.1.3 Edge based detectors

In 1986, Canny introduced the named Canny edge detector [Canny (1986)] which is still widely used nowadays in computer vision.

According to [Canny (1986)], there are three fundamental criteria for a good edge detector:

- Good detection: the edge detector should detect as many real edges as possible.
- Good localization: the detected edge locations should match to the real edges locations.
- Only one response per edge, that is, there should be only one detection for each real edge.

In [Canny (1986)], an optimal function is proposed to detect edges which meets the previous criteria. Furthermore, the article demonstrates that the optimal function can be approximated by the first derivative of a Gaussian. In addition, it is also demonstrated that a point in an edge presents large variation of directions, the idea of the Canny detector is to apply Gaussian derivative filters in the images. Each image is blurred to remove noise and transformed into gradient magnitude and orientation map by the following functions:

$$m(x,y) = \sqrt{I_x^2 + I_y^2}$$

$$\theta = arctan(\frac{I_y}{I_x})$$

where $I_x = G_\sigma^x * I$ and $I_y = G_\sigma^y * I$, and G is the first derivative of a Gaussian and I the original image, $m(x,y)$ indicates the gradient magnitude at position (x,y) in the image, and θ its gradient angle which normally is rounded into four directions: 0, 45, 90, 135. The variance σ used in Gaussian blur function is the parameter that helps to control the number of detected edges. If the smoothness parameter σ is large, the image becomes more blurred, and some real edges might be missed. This breaks the first criterion described before. On the other hand when using a small σ, noisy edges may remain in the smoothed image which breaks the third criterion. Intermediate values of σ should be selected.

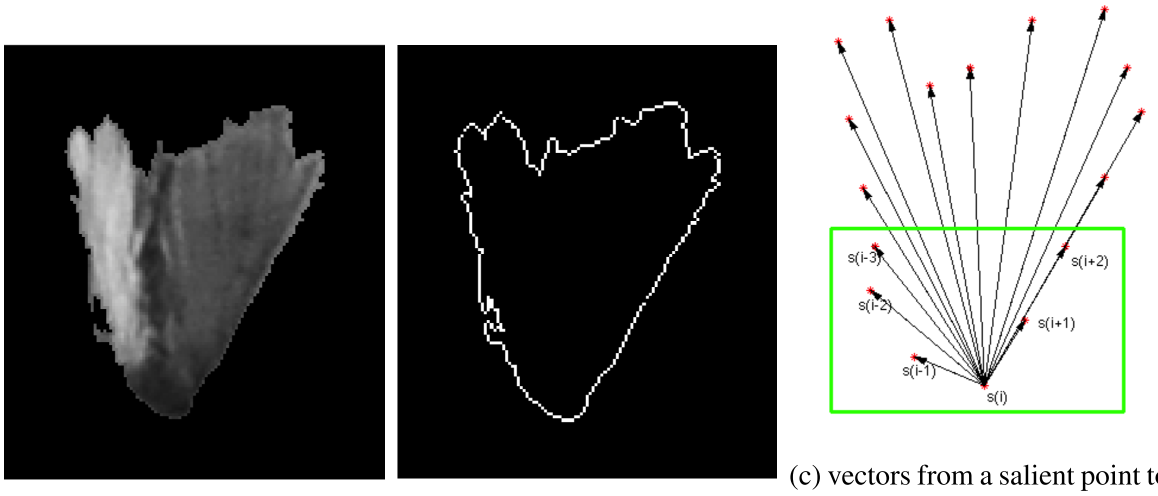
Once the image is blurred, gradient magnitudes of each of four rounded directions at each point are calculated. The direction with the maximum gradient magnitude is selected as the gradient direction of that point. To decide if a point belongs to an edge or not, we should verify if the gradient magnitude of the maximum is above a threshold t. This parameter is similar to the σ variable used to control smoothness of the filtered image. The parameter t controls the number of selected edge points. The value of this parameter is critical since a large threshold may break the edges and make them seem discontinuous. On the other hand, small thresholds may broke the third fundamental criterion producing several parallel lines for a single edge. To alleviate this difficulty, it is proposed to use thresholding with hysteresis. The parameters are formed now by two thresholds t_1, t_2 which are low and high respectively ($t_1 < t_2$). These thresholds are responsible to decide if a point belongs to an edge:

- If the gradient intensity of a local maximum is below t_1, then it is discarded as a point on an edge.
- If the gradient intensity of a local maximum is above t_2, then it is accepted as a point on an edge.
- Otherwise, the point is considered as part of an edge only if it is in the middle of the edge.

These criteria allow us to have a very robust response. As we point out previously, Canny edge algorithm is still one of the state of the art edge detector and it can be used into edge based descriptors.

2.1.4 Beam angle statistics

Beam angle statistics (BAS) is a contour based descriptor used to describe contour of an object. In [Arica and Vural (2003); Arica and Yarman-Vural (2003)], this descriptor is used to represent two-dimensional object silhouettes. This is done by a histogram in which angles formed by neighbouring points around a saliency of the contour are accumulated discretely. With this method, global shape information of an object is retained by the entire contour descriptor.



(a) Original example (b) Extracted contour (c) vectors from a salient point to its neighbours

Figure 2.4: An example which illustrates the procedure to generate beam angle feature (A neighbours sampling is done here just for clarity). Here the saliency is s(i), the first included pair of neighbours into a K-curvature function are s(i-1) and s(i+1).

Beam angle descriptor uses the object boundary in a way such that it is invariant to translation, rotation and scale. The descriptor construction procedure used here is simple: at each saliency point, the boundary curve of an object is extracted and the salient points of the boundary are picked to construct the descriptors.

The angles formed by the line connecting the salient point with its k^{th} right neighbour and line connecting the salient point with its k^{th} left neighbouring point on the contour define the K-curvature function, see figure (2.4). For each of these salient points, a K-curvature function is obtained and a histogram is then accumulated by the outputs of this function. This is, if we have a K-curvature with K=5, five pairs of nearest neighbours are picked in each direction of the saliency point and the angles formed by them are calculated; The output angles are then accumulated in a descriptor histogram. These angles are discretized into 36 bins each one covering a range of 10 grades. This is done for all saliencies on the extracted contours of an object and each salience forms a descriptor. The number of extracted descriptors for each contour is same as the number saliencies in that contour.

For an object in a new image, its contour is extracted and each one is described by the the K-curvature function described before. The similarity between two Beam angle descriptor can be computed using euclidean distance.

2.1.5 Summary

In this section, we have presented some image features which can be applied in computer vision area. There are mainly two groups of features: patch based and shape based features.

Each one of these group of features works well with some specific conditions, sometimes the combination of some of them is necessary to get optimal recognition system. We think that shape or contour based object detection algorithms should perform better in textureless objects than appearance based algorithms (Patch based). On the other hand, patch based features should work better in situations where the detecting object contains rich texture information. In [Opelt *et al.* (2008)], Opelt used both appearance based and shape based methods to detect centers of objects which proved this complementation sometimes is necessary in image recognition. Their experimental results also show that selection of features (preference for shape over appearance or vice-versa) is category specific.

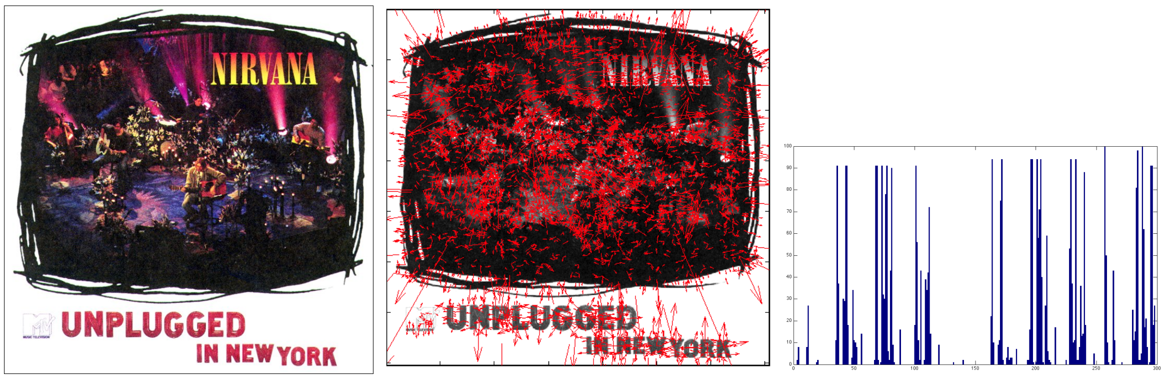
Aside from all previous introduced and other image features, other tools may be needed to complete a object recognition task and give the recognition agent sufficient intelligence to do so. Normally, a proper machine learning algorithm or spatial information treating procedure is indispensable for good performance recognition system. In the following sections we give some references about these tools.

2.2 Image classification

The goal of an automatic image classification system is to categorize images based on their content. In order to obtain a good performance the system is trained using a set of labelled images. These training images with their class labels normally present some challenges for the classifiers, such as: intra-class variation, cluttered scenes, partially occluded objects, light variation, objects from different viewpoints. The objective of the classifier is to capture this variability and the common characteristics of the objects and use them to categorize new images. Image classification can be applied in a wide variety of applications such as Content-based image retrieval, landscape or scene recognition, etc.

In this section we describe some state-of-the-art machine learning tools for image classification.

2.2.1 Bag of words



(a) Original image of an album cover (b) Detected SIFTs with different size and orientation (c) Encoded histogram of the original image

Figure 2.5: An example which encodes an image into a histogram of occurrences. The size of the image descriptor (histogram) depend on the size of the codebook, in this case: 300

In [Csurka *et al.* (2004); Willamowski *et al.* (2004)], the bag of words (BOW) method is presented in the computer vision area. The method is inspired from text retrieval and text categorization. This approach has demonstrated to be useful in image categorization too. The idea is to represent an image using a bag of visual words. Visual words are patches on the image which represent some kind of interest information related to the image features (the color, shape or texture or low-level feature descriptor like SIFT). In [Csurka *et al.* (2004); Willamowski *et al.* (2004)], the authors used SIFT descriptor as the visual word in their experiments. A bag of visual words is a sparse vector of occurrence counts of visual words which can represent the information about the meaning of the image. Thus BOW approach is a method to transform each image to a characteristic representation of the image or counting of the different parts of the image.

However, for a high dimensional feature space such as SIFT descriptors, words contabilization may be computationally unfeasible. A low length codebook is then needed to reduce the dimensionality of the problem. The BOW model includes in general the following steps:

- Feature representation.
- Codebook generation.
- Image representation.

In the original work [Csurka *et al.* (2004)], feature representation is carried out using the SIFT algorithm to detect and describe keypoints of images. The result representations of the image characteristics are 128-dimensional vectors which are invariant to intensity, rotation, scale and small affine variations.

Codebook creation implies a clustering process in which similar feature descriptors are grouped into same cluster. Each cluster center corresponds to a visual word or common aspect of the images. Each visual word is an entry of the codebook. K-means clustering is one commonly used algorithm to create an efficient codebook although other clustering techniques are used too, like: Hierarchical K-means clustering [Mikolajczyk (2006)], Agglomerative clustering [Leibe and Schiele (2003)] or Gaussian Mixture Models (GMM) [Wikipedia (2004d)]. In K-means, k cluster center positions are randomly sampled in the beginning. The clustering process of K-means is an iterative algorithm [MacKay (2002)] which contains two parts:

- Assignment step: in this step, each image feature is assigned into a cluster by finding the cluster with minimum distance between its center and feature descriptor.
- Update step: After all feature descriptors are assigned, the center of each cluster has to be updated by calculating its position as the mean of all points assigned to the cluster in the previous step.

The iterative process is finished when any of the following criteria is met: No actualization of the cluster centers is performed, or the maximum number of iterations is reached. Some disadvantages of K-means must be pointed out: The number of clusters K has to be set beforehand and the initialization of clusters positions is done randomly. This means that, frequently the iterative process converges toward a local minimum. This problem can be solved by running the iterative process various times with different initializations. However, it makes the selection of the optimal k computationally expensive. However, this parameter does not greatly affect the final performance of BOW given that a large K

(~`1000-10000` visual words) is selected.

In problems with a large corpus of images, like a content based image retrieval system (CBIR) [Wikipedia (2004a)], the above clustering process may become a computational bottleneck. A solution to this problem is to randomly sampled feature points [Li *et al.* (2009)]. Each of these points constitute a codebook entry. In test, each feature is assigned to a entry if the distance between the feature and the entry is small than a threshold. More than one entry could be assigned for each feature. The main idea is to replace a clustering generated codebook by a set of generated hyper-spheres. This technique is demonstrated to be efficient to replace the clustering process.

Using the generated codebook (a set of different cluster centers), each image can be described directly as a histogram of number of occurrences through the codebook. This is done by assigning each image SIFT descriptor to its most similar cluster in the codebook. The numbers of allocations of each visual work in the codebook forms the final histogram. This histogram is a description or counts of the characteristics of the image.

After all images are transformed into histogram vectors, we can then use them and their corresponding labels to train a model and use the model to classify new images. Almost every classification algorithm can be used. For example, in the original Willamowski's work [Willamowski *et al.* (2004)], both Naive Bayes and SVM are used in their experiments which give good results. It has to be mentioned that despite of the simplicity of Naive Bayes, actually it is widely applied in document classification in information retrieval field.

To classify a new image, first, its SIFT descriptors are extracted. Then a histogram is created by using the codebook previously generated which summarizes the characteristics of the image. Finally, the trained classifier determines the class label of the histogram.

2.2.2 Supervised codebook creation

One criticism that is generally done to BOW is that the clustering process is unsupervised. Meaning that the image labels are not taken into account when building the clusters. This might lead to non discriminative clusters and to an overall deterioration of the classification system. There exist many papers that propose to apply different semi-supervised techniques to obtain the codebook [Moosmann *et al.* (2008); Lepetit *et al.* (2005); Martínez-Muñoz *et al.* (2009); Moosmann *et al.* (2007); Winn *et al.* (2005); Sukthankar2 (2008); Zhang and Dietterich (2008)]

One of most used supervised algorithm in feature codebook creation is Random Forests

[Statistics and Breiman (2001)] which allocates similar image features in the same leaf for future retrievement. This classification algorithm is the basis of some of the algorithms presented below. The main idea of Random forests is to construct an ensemble of classifiers where each decision tree is trained using a subset of examples and a subset of attributes for each node division. The learning process consists of following steps Algorithm (1):

Algorithm 1 Random Forests

```
1. A set of $m$ decision trees are generated.
2. For each tree, a training set is selected by choosing n samples with replacement from
   the training examples of size $n$.
3. When training the tree using the selected training set, for each node of the tree, a subset
   of $s$ attributes ($s$ is much smaller than the number of attributes) are chosen as candidate
   split variables. The best split test is used. A typical criterion used to determine the
   quality of the test is the information gain measure or the Gini criterion. Each tree is
   grown until if all feature samples belong to the same class or all attributes of these
   samples have same values.
```

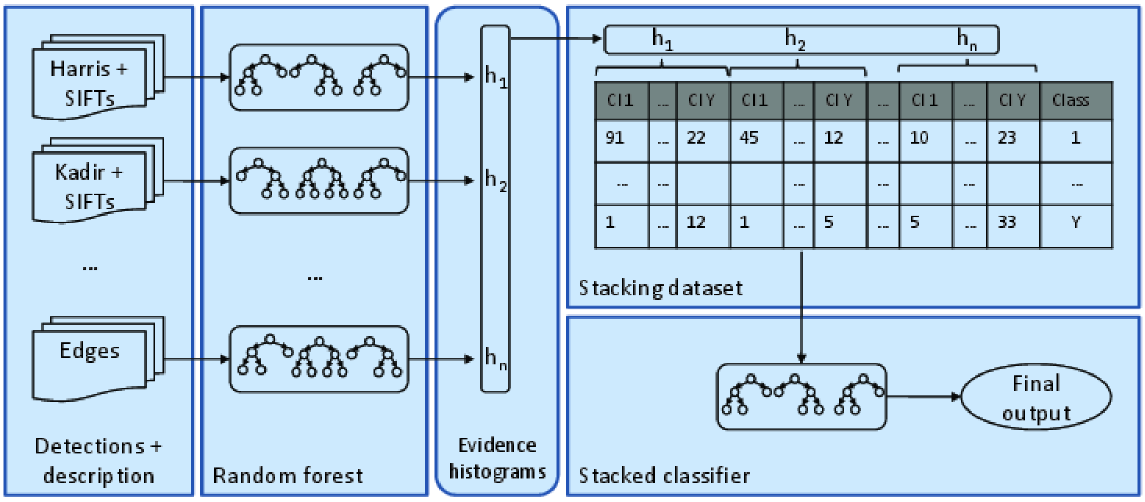
Random Forests is a very accurate classification algorithm over a wide variety of problems [Caruana and Niculescu-Mizil (2006)]. In addition, it is a computationally effective algorithm on large databases.

One of the image classification algorithm based on Random forests is [Moosmann *et al.* (2008)]. In [Moosmann *et al.* (2008)], Moosmann proposed Extremely Randomized Clustering Forests (ERC-Forests) which achieves good performance. This algorithm is based on an extremely randomized Random forests that are very quick to be built. Another contribution of the work is the use of ERC-Forests as spatial partitioning method to cluster image feature descriptors instead of using the trees to classify images directly like in other previous works [Lepetit *et al.* (2005)].

The ERC-Trees are built recursively top down. In each division t, the division criterion corresponds to a binary test in which the image features are split into two sets. The division criterion is a binary test which consists to a comparison like $T_t = f_i(t) \leq \theta$, where $f_i(t)$ is a randomly selected attribute. The split threshold θ is also selected randomly. To measure the quality of a test, Shannon entropy is used:

$$S_c(C,T) = \frac{2I(C,T)}{H_C + H_T}$$

where H_C is the entropy of the class, H_T the entropy of the partition after the division and $I(C,T)$ the mutual information. The randomly selected split is considered valid if $S_c(C,T)$ is above a threshold S_{min} or if after T_{max} attempts none of the selected splits has



Cl 1	...	Cl Y	Cl 1	...	Cl Y	...	Cl 1	...	Cl Y	Class
91	...	22	45	...	12	...	10	...	23	1
...	...			...		...		...		...
1	...	12	1	...	5	...	5	...	33	Y

Figure 2.6: The architecture of the image classification system proposed in [Martínez-Muñoz *et al.* (2009)]

a $S_c > S_{min}$, then the split with best Shannon entropy is selected. The parameters S_{min} and T_{max} allow us to control the strength and randomness of the generated trees.

The trained trees transform each feature descriptor into a set of leaf node indexes (one index per tree). Then the votes for each index in the trees are accumulated and form a global histogram for the image.

The remaining part of the algorithm is similar to the BOW approach used for image classification. In training, all the histograms of images are used to training a classification model. In test, first the image feature descriptors are extracted, then these descriptors are used with the trained ERC-Forest to construct an image histogram. The histogram is then passed through the classification model to give a prediction of the image label.

Another algorithm based on Random Forests is [Martínez-Muñoz *et al.* (2009)]. In [Martínez-Muñoz *et al.* (2009)], the authors propose to use stacked evidence trees. In their work, they use the standard Random Forests as the evidence trees that stores a histogram of the number of training features from each category that reached each leaf. Figure (2.6) shows the overall architecture of the system. In the first stage, feature detection and description are done, both keypoint and edge based features are used. This algorithm is thought to combine multiple image features. So for each image, a set of pairs of detector and descriptor c is applied to the image and a bag of feature descriptors is produced $B_I^c = X_{I,1}^c, ..., X_{I,N1}^c$. Then those feature vectors are dropped into the previous trained classification system. The learning process involves three steps:

- Learning a Random Forests: The training data used in each decision tree are obtained by using those feature vectors of a training image subset in which each image is selected randomly. Then in each node of a tree, a binary division criterion $x^c_{I,j}[a] \leq \theta$ is created by selecting randomly an attribute a from a subset of attributes (with size $1 + \lfloor \log A \rfloor$) and a threshold θ. Branch left if the output of the test is true, right otherwise. The tree stops the growth when the number of examples of each node is less than 20.

- Constructing the second-level training set: Because each tree is grown by a subset of images, then exists out-of-bag images that were not used to construct each tree. To construct the second-level training set, the feature vector of each training image I is passed through each trees for which I was out-of-bag. Then the histograms are accumulated and normalized. These are called evidence histograms. This gives us a unique histogram for each feature Random Forests and a global feature vector is created by concatenating each of these histograms which will be used as training sample in the stacked classifier.

- Learning the stacked classifier: In the stage, an ensemble of 200 decision trees are trained by using Adaboost algorithm.

In the test phase, each feature descriptor is passed through the Random Forest and the evidence histograms are generated. The output evidence histograms are normalized and concatenated to form a global feature vector. This feature vector is passed into a classifier in the second layer of the architecture which will give us the final label of the image.

This algorithm performs well in classifying very similar objects like different species of stoneflies and for generic object categorization, the classification performance of their method is shown in Figure (2.7) with PASCAL 2006 dataset.

In our approach, we are extending this work to use Random Forests to learn training evidence with their displacements to the center of a detecting object, The learned Random Forests will be used to predict the displacements to the center of an object for future feature descriptors.

2.2.3 Spatial pyramids matching

If there is a drawback in codebook based methods is that the global and image structure information are missed once feature descriptors are clustered and orderlessly quantized.

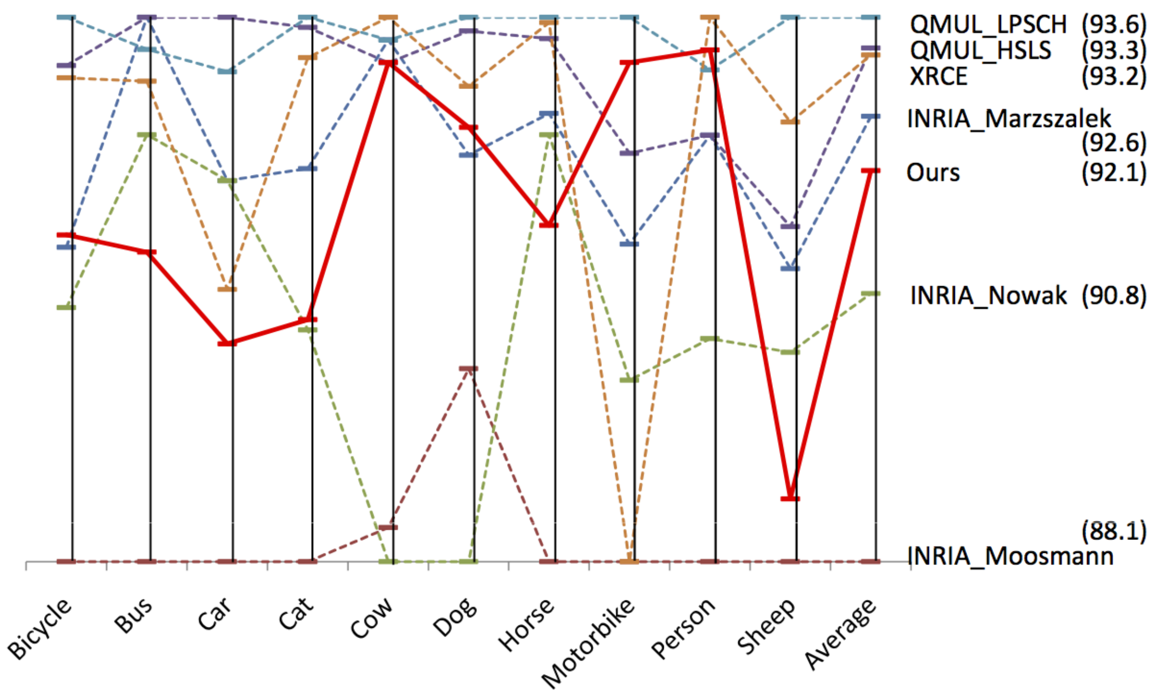


Figure 2.7: Classification performance comparing with other state-of-art algorithm of the approach in [Martínez-Muñoz *et al.* (2009)]

This deficiency may cause false positives in matching problems where different contents could be concluded to be the same [Savarese *et al.* (2006)]. Since this method is based on identifying "parts" of objects, the result is not guaranteed to be the desired object. For example, detecting a wheel does not guarantee that there is a car in the image. This problem also occurs in document classification based on BOW [Lijffijt *et al.* (2011)]. In order to overcome this problem, in [Lazebnik and Schmid (2006); Yang *et al.* (2009)], the authors use a spatial pyramid matching method which creates a hierarchical histogram with different weights in each sub-histogram according to the contributed global structure information. The same idea is used in [Bosch *et al.* (2007)] where the authors use spatial pyramid kernel to represent shape features. In their work, they extend the original version of using static histogram weights in each level, they use an extra validation dataset to learn the weights of each level which improves the classification performance. Other authors added more geometric constraints in feature points matching problems by using Random Sample Consensus (RANSAC) algorithm to eliminate wrong matching points [Wei *et al.* (2008)]. The idea of using spatial information is applied in our approach too, we propose to use relative distances between the SIFT feature points and the center of the detecting object. These distance information are stored in training and are recovered in test to vote the center of an object in a test image.

As described before, the main idea of this algorithm is to use the spatial location of each detected feature by adding spatial constraints. To do so, once the codebook is created, each image is decomposed in multiple levels to create a hierarchical feature histogram. In each level, the image is segmented and separated into different regions. Each region is used to create a sub-histogram according to the previously constructed codebook and the images features that are located in the region. The number of counts of each type of image feature is stored along with its corresponding level and grid index as a feature vector. This is shown in figure (2.8).

To measure the similarity of two images or features vectors X, Y, a histogram intersection function is used to find the common image features of these images:

$$I(H_X^l, H_Y^l) = \sum_{i=1}^{D} min(H_X^l(i), H_Y^l(i))$$

where $H_X^l(i)$ and $H_Y^l(i)$ are the numbers of a specific image feature vector from X and Y that fall into the ith cell of the grid of level l.

As proposed by the authors, the weight of each $I(H_X^l, H_Y^l)$ should be proportional to

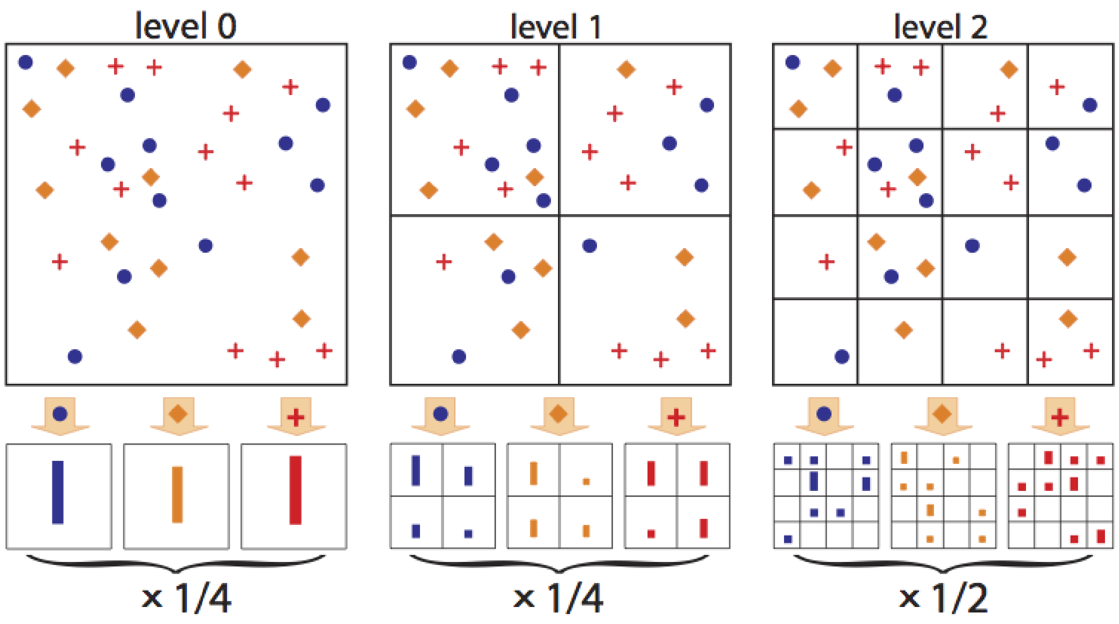


Figure 2.8: There are three levels with a codebook of three types of image features. In each level, the image is divided in different grids, and feature of each type is counted, the number in the bottom indicate the weighted of each sub-histogram (An example extracted from [Lazebnik and Schmid (2006)])

the width of the level l. So the pyramid match kernel is:

$$k^L(X,Y) = \frac{1}{2^L} I^0 + \sum_{l=1}^{L} \frac{1}{2L - l + 1} I^l$$

where I^l is the abbreviation of $I(H_X^l, H_Y^l)$.

The sum of the histogram intersections is the score of similarity between two images. This is calculated by applying pyramid match kernel function for two images:

$$K^L(X,Y) = \sum_{m=1}^{M} k^L(X_m, Y_m)$$

where M means the total number of image features in the codebook.

In the original work, multi-class classification task is done by using a support vector machine (SVM) trained using one-vs-all rule: There is one model for each class to separate its samples from the rest. Spatial pyramids matching kernel is then used as the kernel of the SVM. This algorithm performs well in classifying images with structures like scene categorization

2.2.4 Naive-Bayes Nearest-Neighbour

Codebook is a form of nearest neighbours search which is a method that compresses the information in return for a good computation efficiency in test. In the sense that each test descriptor is assigned to the nearest codebook entry. If the image sample is small, the required precision of retrievement in test could be more strict. Therefore, direct nearest neighbour search is sometimes needed to get the closest features. In [Boiman *et al.* (2008)], the authors claimed that the creation of a codebook loses feature information in the compression process of image features. In addition, the authors also defend the non-parametric methods which does not lose any feature information, because there is not a compression process in these models.

In their work [Boiman *et al.* (2008)], the authors give an explanation for the poor performance of nearest neighbours applied to image classification that previous works had reported. These works used the "image to image" distance instead the "image to class" distance to label an image. In the "image to class" distance approach, the training feature descriptors are saved for each class. In a test image, the distances between each test feature descriptor and the most similar training feature descriptors of each class are calculated. All these distances are accumulated for each class. Finally, the class with minimum distance will be selected as the label of the test image. Despite of its simplicity, (That is why they named the algorithm Naive-Bayes Nearest-Neighbour (NBNN)) the algorithm approximates the theoretical optimal Naive-Bayes classifier [Boiman *et al.* (2008)]. The NBNN algorithm is summarized as follows in algorithm (2).

Algorithm 2 NBNN

```
1. Extract descriptors $d_1$,...,$d_n$ of the query image $Q$.
2. $\forall d_i\ \forall C$ compute the Nearest neighbour of $d_i$ in C: $NN_C(d_i)$
3. $\dot{C} = argmin_C \sum_{i=1}^{n} \|d_i - NN_C(d_i)\|^2$
```

This algorithm does not require training, the neighbours retrieved in test can be computationally costly. However, in [Boiman *et al.* (2008)], they used KD-trees [Bentley (1975a)] to save the training image feature descriptors for each class. This reduces the retrieval complexity to $O(NlogN)$ for each image feature. And the total time complexity for one query image is: $O(n_C * n_D * log(n_D))$ where n_C is the number of classes and n_D

is the number of descriptor searches.

2.3 Object detection

Object detection is another important subarea in computer vision due to that many possible applications can be developed with a good object detector: face detection [Viola and Jones (2004)], pedestrian detection [Gall and Lempitsky (2009)], video surveillance [Dedeoglu (2004); Garcia-Martin and Martinez (2010)], etc. The objective of object detection is to determine the presence of a specific object and its location in a query image. Here we present some state-of-the-art algorithms in object detection area which inspired us in our own object detector.

2.3.1 Sliding Window based

Sliding window approach is an intuitive and widely used method that can be applied to object detection. The approach involves scanning the image with a fixed-size rectangular window and the inner region acts as the entry to a detector. A detector will decide if an object will be in the region o not. In this way, the detector model only works with a subset of the original image which can simplify the model complexity. With the Sliding window approach, a template matching method usually is implemented to decide if the region in the window contains a specific object. The template defines the shape of the sliding window and other geometric constraints for the object to be detected. In [Schneiderman and Kanade (2000)], the authors presented a statistical model in which each window has a statistical probability of containing an object. The features used are visual attributes to estimate the probabilities and each model is trained for a specific orientation of an object. In [Breazeal and Scassellati (1999); Anderson and McOwan (2006)], the authors use template based algorithm to detect faces. In [Dalal and Triggs (2005)], the authors extend templates to build features from grids of Histograms of Oriented Gradient (HOG) to detect pedestrians. This algorithm gives very good results.

The benefit of using templates matching approach is that in each template, the geometric constraints are set. This method has the advantage with respect to BOW that it detects objects as a whole which reduces the number of false positives. Because in BOW based methods, the feature quantization is done disorderly to generate a histogram for each image, this makes it possible that two images with different structure or scene generate a similar histograms. On the other hand the same benefit may cause undesirable results: Some minimum affine transformation changes on the detecting object, which occur frequently in real world problems, can make it unfeasible for the template to indicate a positive case.

Despite of the simplicity of the sliding window approach, normally it is a computational inefficient algorithm, because the performance of the algorithm depends on the scale step and move distance taken each time by the window. Big steps yields bad performances and less time consuming, otherwise, small steps may assure the performance of the detector by checking each possible subregion, but the high number of windows verifications makes the detector unfeasible for a real-time application. Some authors prefer to reduce the computational cost by searching only important subwindows in which detections are performed. For example in [Lampert *et al.* (2008)], the authors used Branch-and-Bound Search to reduce the number of candidate regions to be searched, the inspections of the presence of an object will be done only in those windows with high scores calculated with a bound quality function.

Viola-Jones object detection framework

One of the state-of-the-art and most utilized algorithm based on sliding window is proposed by Viola and Jones in [Viola and Jones (2004)]. This algorithm solves the mentioned computational problem using fast image feature calculation and computational optimized subwindow verification. Its popularity is due to its performance and speed in object detection. In this algorithm, a modified version of Adaboost is implemented: a stack of boosting-like detectors are created. The false positives are detected in early stages. The authors also proposed to use Haar-like features with integral images techniques to cut the computational cost and to make the detection algorithm useful in real time application.

Haar-like features are used in this work as the base image detector. A Haar-like feature is defined as the difference of the sum of pixels of certain area inside a rectangle region of an image. Viola and Jones defined two, three and four rectangles feature as shown in Figure (2.9). Posteriorly, Rainer Lienhart improved this approach by introducing rotated Haar-like features which enrich the feature variety and improved the performances of the algorithm.

To compute the difference between two rectangle sums quickly, the computation are performed using integral image arithmetics. Integral image is a transformation of an image where each pixel is assigned the sum of the values of all pixels in the original image to the left and above its position. This is:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

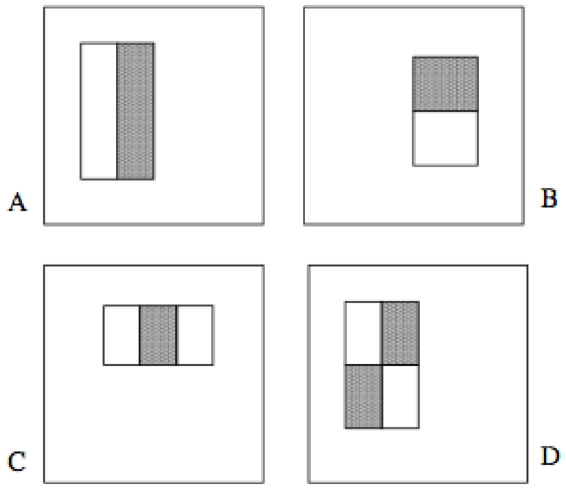


Figure 2.9: Haar-like features examples

where $i(x', y')$ indicates the pixel value of the position (x, y) in the original image. With the transformed integral image, the calculation of any inner rectangle can be done in few operations. See an example in figure (2.10). The sum of pixels in the rectangle in gray can be computed as: ii(C) + ii(A) - (ii(B) + ii(D)).

Haar-like features are weak features, in the original work, these features are applied with 1-level decision tree: For each feature, a comparison between a randomly selected Haar-like rectangle difference and a random threshold is done. After that, they apply a feature learning process to select 1-level trees obtained in the previous step. A modified version of Adaboost [Freund and Schapire (1997)] is used to select the most promising weak classifier, see Algorithm (3).

The selected classifiers are successively appended to a cascade of classifiers and for each classifier a threshold is learned to minimize false negatives. In test, the query subwindow is processed sequentially through the cascade of classifiers until it is rejected or passes all stages. If the query subwindow is rejected by any classifier in the cascade of classifiers, no further inspection is performed in the following weak classifiers. In this way, negatives examples can be rejected in early stages and the overall inspection time is reduced. The defined Adaboost variation guarantees to put the most important classifiers in early stages. As it is shown in their work, using only the first two classifiers can achieve a false negative rate of approximately 1% and a false positive rate of 40% in a face detection problem. The same algorithm is used in [Levi and Weiss (2004)]. In this study it is shown the importance of learning discriminative features when the training dataset is small.

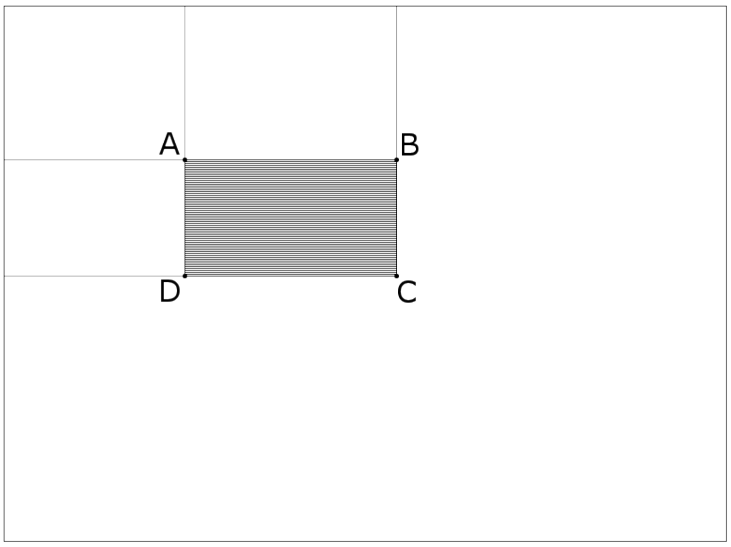


Figure 2.10: An example of sum of the pixels in a rectangle by using integral image.

2.3.2 Contour based object detection

Another object detection algorithm is proposed in [Opelt *et al.* (2006)]. This is based on edge detection. Detected edges may be sometimes noisy, because many of them do not belong to the object of interest like background edges. Therefore, a learning process is required to find the most discriminative ones to be used in the model. In [Opelt *et al.* (2006)], the authors proposed to use Boosting based learning algorithm (similar to the

Algorithm 3 Feature selection

```
1. given examples ($x_i$, $y_i$) with $y_i = 0, 1$ for negative and positive cases.
2. Initialize weights equally for every examples $w_1, i$
3. for $t = 1...T$ do
4.     Normalize the weights of examples: $wt, i = \frac{w_{t,i}}{\sum_{j=1} w_{t,i}}$
5.     For each feature, train a classifier $h_i$
6.     Select the classifiers $h_j$ with the lowest error $e_j = \sum_i w_j |h_i(x_i) - y_i|$
7.     Update each example distribution: $w_{t+1,i} = w_{t,i} B_t^{1-e_i}$
8.         $e_i = 0$ if $x_i$ is correctly classified,
9.         $e_i = 1$ otherwise, $B_t = \frac{e_t}{1-e_t}$
10. end for
```

Adaboost based method in Viola-Jone's work) to select discriminative combinations of boundary fragments to form a Boundary Fragment Model (BFM) detector. In this method, a codebook of similar positive curves is created for each object for future retrievement. The use of a voting map is also done in our approach to find the position of the center of an object. A similar approach is applied in Shotton's previous work [Shotton *et al.* (2005)]. The main difference between these two methods are:

- The level of segmentation required in training;
- The number of fragments used to construct each weak detector.
- The center detection methods used in both algorithms.

Additionally, in another work [Ferrari *et al.* (2007)], the basic contour based object detection is extend to detect the exact object boundaries in cluttered images and a class-specific shape model is proposed to handle object deformation and large intra-class variability.

In [Opelt *et al.* (2006)], each edge is detected by Canny detector. Firstly, a Boundary Fragment Alphabet is built by selecting suitable candidate boundary fragments: (i) fragments with high probability to be in positive images; (ii) fragments that have a good localization of the centroid of the object in the positive image. The candidate boundary fragments are selected from the contours of positive training images. For each contour, three segments started from three randomly chosen locations on the contour are generated. Agglomerative clustering is used to group all selected segments and to form an alphabet. The Chamfer distance [Borgefors (1988)] is used in their method to measure the similarity between edges. For each cluster, the displacements from the segment to the centroids of objects are recorded.

Once the alphabet is generated, weak detectors are created by combining two candidate edges which agree in their centroid estimates. For a weak detector h_i, its output on an image I is:

$$h_i(I) = \begin{cases} 1 & \text{if } D(h_i, I) < th_{h_i}, \\ 0 & \text{otherwise} \end{cases}$$

$$D(h_i, I) = \frac{1}{m_s^2} \cdot \sum_{j=1}^{k} distance(\gamma_j, I).$$

where $distance(\gamma_j, I)$ is the distance to the best matching edge chain in the image. m_s is a penalty applied if the centroid decision of two selected edges are not overlapped. The learning process is used to select promising weak classifier and learn the thresholds

th_{h_i} by using Adaboost. In test, the algorithm (4) is used. First, edges are extracted from the test image, then these edges are checked in the trained weak classifiers, those edges that passed the test of a weak classifier can cast a vote to the center of the object using the displacements stored in the corresponding passed weak classifier. After that, the mean-shift is used for the accumulated vote map to obtain the center of the object. Finally, back-projection is used to segment the detected object region from the test image.

Algorithm 4 BFM algorithm used to test an image

```
1. Extract edges from the query image
2. Test each weak classifiers $h_i$, if $D(h_i, I) < th_{h_i}$ the displacements saved in the training
   is used to vote object centroid in Hough space.
3. Use mean-shift to estimate the center of object.
4. If the center response is above a threshold $t_{det}$ then means the object is detected in that
   position.
5. Back-project the fragments that voted for the center and it is also used to segment
   object from the image.
```

2.3.3 Hough Forests

A second family of detection algorithms are based on the Generalized Hough Transform (GHT) [Leibe *et al.* (2004); Opelt *et al.* (2006); Wolfson (1991); Gall and Lempitsky (2009); Gall *et al.* (2011)]. In [Gall and Lempitsky (2009); Gall *et al.* (2011); Leibe *et al.* (2004)] they apply a GHT voting map to detect an object position which is used in our approach too. The Hough Forests proposed in [Gall and Lempitsky (2009); Gall *et al.* (2011)] is a state of the art algorithm which achieves very good results in object detection. Another contribution of this work is that their authors adapted GHT into a Random Forests in which voting can be handled efficiently.

Hough Forests is basically an extension of Random Forests for computer vision area. The Hough Forests uses HOG patches as the image feature and adapts the idea of Random Forests to split the feature data in different leaves. Hough Forests has the following advantages according to [Gall and Lempitsky (2009)]:

- The set of leaf nodes can be used as a class specific discriminative codebook where similar features are located in same leaves. In this way, the voting performance can be guaranteed.

- It can be trained on large corpus of data, the retrieval process in test is very efficient.

- This method can tolerate noisy labels and errors in the training data.

In the original works [Gall and Lempitsky (2009); Gall *et al.* (2011)], each Hough Forests tree is built based on patches $P_i = (I_i, c_i, d_i)$, where I_i is the descriptor of the patch, c_i is the class label of the patch and d_i, a two dimensional offsets (in x and y) from the patch to the center of the object. All training instances need to be labeled. In their work, each patch consists of C feature channels of 16-by-16 pixels rectangles, these feature channels are defined as follows: three color channels of the *lab* color space [Wikipedia (2004b)], the absolute values of the first-order derivatives $\frac{\alpha}{\alpha x}$, $\frac{\alpha}{\alpha y}$ and the second-order derivatives $\frac{\alpha^2}{\alpha x^2}$, $\frac{\alpha^2}{\alpha y^2}$ and nine HOG-like [Dalal and Triggs (2005)] channels. To increase the invariance under noise, the authors applied the min and the max filtration with 5-by-5 filter size, yielding C = 32 feature channels (16 for the min filter and 16 for the max filter).

During training, the nodes split the patches in two sets according to their appearance. This is done by applying a binary test that compares a threshold value τ with the difference of two intensities at two randomly selected locations in the patches, (p, q) and (r, s), for a randomly selected appearance channel α.

$$t_{\alpha,p,q,r,s,\tau}(I) = \begin{cases} 0 & \text{if } I^\alpha(p,q) < I^\alpha(r,s) + \tau \\ 1 & \text{otherwise} \end{cases}$$

where $t_{\alpha,p,q,r,s,\tau}(I)$ is the test, and $I^\alpha(x, y)$ is the intensity value at location x and y of the channel α. This test is done various times with different values selected at random. The one with best score is selected as the split criterion for that node. In their work, the score is measured by two factors:

- The impurity of class labels c_i:

$$U_1(A) = |A| \cdot Entropy(c_i)$$

$$\text{where } Entropy(c_i) = -c \cdot logc - (1-c) \cdot log(1-c)$$

- The offset uncertainty:

$$U_2(A) = \sum_{i:c_i=1} (d_i - d_A)^2$$

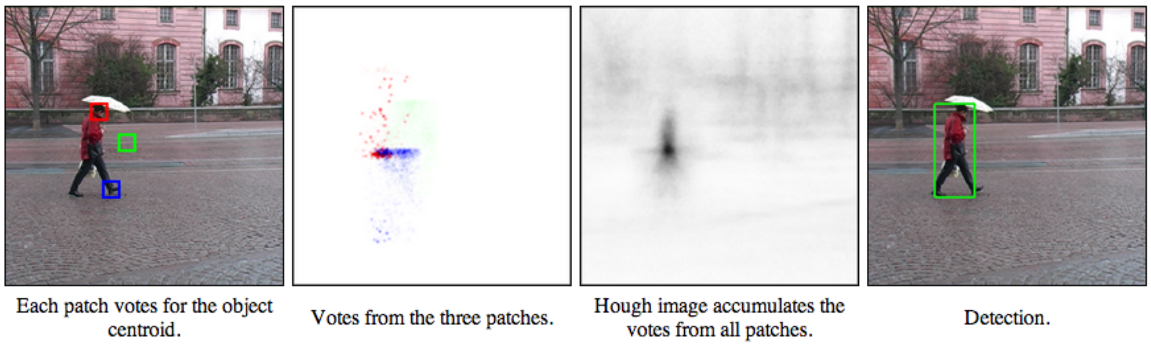

Each patch votes for the object centroid.
Votes from the three patches.
Hough image accumulates the votes from all patches.
Detection.

Figure 2.11: An example of voting process in Class-specific Hough Forests

where A is the set of divided patches. The objective is to minimize the impurity and uncertainty respectively. The best division criterion U_* is selected by:

$$argmin_k(U_*(p_i|t^k(I_i) = 0) + U_*(p_i|t^k(I_i) = 1))$$

The best criterion U_* depends on the selected measure used to calculate the score. The authors use the measures $U_1(A)$ and $U_2(A)$ randomly unless the number of negative patches is small (smaller than 5% of total examples), in that case, U_2 is selected.

Patches are stored in the leaf nodes to use as votes in the test. The authors adapted the idea of voting map and used the displacements saved previously with each image features to ”recover” the center position of an object. Figure (2.11) shows an example of a vote map in which the accumulated votes concentrate in the center of a pedestrian. The size of the pedestrians are determined by a fixed rectangle. To detect pedestrians in different sizes, the authors scale the test images in multiple sizes and perform the detection always using 16-by-16 pixels patches.

Chapter 3

Proposed algorithm

The main objective in a object detection task is to detect the position and the size of the object. To determine the location of an object is enough to detect its center. However, for a full localization of the object in the image, it is also necessary to estimate the region that the object occupies. The proposed method achieves this in two steps. First, the center position is detected (detection step) and then its size is estimated (size estimation step).

The proposed method is based on voting accumulated evidence using Random Forests or KNN. Previously, several methods use accumulated voting to detect object locations [Leibe *et al.* (2004); Opelt *et al.* (2006); Wolfson (1991); Gall and Lempitsky (2009); Gall *et al.* (2011)]. One limitation in these algorithms is that they do not handle object with different orientations along the axis perpendicular to image plane (z-axis). In the proposed approach, we use orientation invariant patch descriptors and vote transformations in order to solve this problem.

The proposed method is based on Evidence trees [Martínez-Muñoz *et al.* (2009)]. In [Martínez-Muñoz *et al.* (2009)], the authors used evidence trees to transform image features into a class histogram of accumulated evidence. This histogram is used as the feature vector for a stacked classifier that outputs the final class of the system. The system was designed for image classification. In [Martínez-Muñoz *et al.* (2009)], it is shown that it performs well for classifying different species of stoneflies (a problem with objects with low inter-class variation) and for generic object categorization (PASCAL 2006 dataset). The approach presented here extends that work to object detection. The key difference between the adapted algorithm and the original one is that the evidence trees of the proposed method store in their leaves the relative distances from image features to the location of the objects of interest instead of the class labels of image features.

The objective of image classification approaches is to provide the correct label for an test image, however, this type of algorithms might leave out geometric information in their model construction in order to improve the computing efficiency. For example, BOW based methods count image feature descriptors and form an orderless histogram regardless the locations of feature descriptors and their possible geometric relationships with other descriptors. Using the above method, we solve this kind of geometric information gap. Hence we can be able to recover the center of an object using relative distances with a voting map. In addition, in this study we also adapted NBNN [Boiman *et al.* (2008)] to achieve object localization.

The relation of the image patch descriptor to use directly influences the performance of the detection algorithm. For example, class specific Hough Forests algorithm [Gall and Lempitsky (2009)] is not able to detect objects if the orientation of the object changes in the test images. This is because this algorithm uses rectangular patches of HOG values which are not orientation invariant. We use SIFT descriptor as our base image patch descriptor. The primary reason is that the SIFT algorithm is able to capture the orientation and scale of the detection. The proposed algorithm uses this information to detect object in different scales and orientations. Hough Forests algorithm estimates the size of the object by the scale of the resized image, because HOG based patches are not scale invariant. In our approach, the size is determined based on the scale of the image patch descriptors, in our case the size of the SIFT region. This is one of the alternatives used to estimate the object size, the same image scaling approach used in Hough Forests can also be adapted in the proposed method.

Most object detection algorithms use both positive and negative images to train the model [Gall and Lempitsky (2009); Viola and Jones (2004)]. In the test phase, a test image patch descriptor will be classified as positive or negative. If the result is positive, then this feature can cast a vote by deciding the location of the center position. However, training with negative examples poses some difficulties. First, it is almost impossible to cover all possible non-object negative examples. Second, the background or non-object patches can be different between the training data and real data from a production environment. In consequence, using negative examples in training may lead to overfitting. In our approach, we tried to avoid this problem by using only positive image features to train the model. Our approach presents the following advantages:

- Handles orientation changes of objects (along the axis perpendicular to the image plane).
- Handles scale changes of objects.
- It is a generic object detection algorithm which can be trained and detect any kind of object.

- No need to be trained with negative examples. The object detection performance is stable on any background scenes.

3.1 Algorithm description

In this section, we give a general description of the proposed algorithm. The entire procedure of our approach is divided in various parts. Here we list the sequence of each submodule used in our method in Algorithms (5,6). In fact, our approach can be easily extended, because some of these submodules can be replaced by other methods: SIFT detector can be replaced by another scale-orientation-invariant feature; different classification algorithms can fill into the ”model” part. After the maps are created for each descriptor in a test image, the center selection is done, the maximum response positions of the maps are used to be selected as the final predictions. But the size of the detected object can not be computed directly. Thus, this decision-making task has to be treated differently for each problem and will be explained in the experiments section.

Algorithm 5 Training

```
Require: positive_cropped_images, annotated_center_positions, descriptor_size
 1. feature_vector = empty
 2. for image in positive_cropped_images do
 3.     detected_locations = mesh_detection(image, descriptor_size)
 4.     [feature_descriptors, angles] = extract_descriptor(image, detected_locations, descrip-
        tor_size)
 5.     annotated_center = get(annotated_center_positions, image)
 6.     feature_displacements = normalized_displacements(feature_descriptors, anno-
        tated_center, angles)
 7.     add(feature_vector, [feature_descriptors, feature_displacements])
 8. end for
 9. model = model_train(feature_vector)
10. return model
```

The images containing the object of interest are used to train the model. For each of these images, the image features are extracted using an appropriate scale depending on the size of the image. The displacements from the location of the feature descriptor to the center of the object are computed. The displacements are rotated according to the orientation of the feature descriptor and are scaled down by dividing the displacements by the size of the feature descriptor. To train the model, the samples are the descriptors with their associated transformed displacements. The objective of the model is to cluster similar

Algorithm 6 Test

```
Require: test_image, feature_descriptor_sizes, model, map
 1. maps = empty
 2. for size in feature_descriptor_sizes do
 3.    map = create_matrix(size(image))
 4.    detected_locations = mesh_detection(test_image, size)
 5.    [feature_descriptors, angle] = extract_descriptor(image, detected_locations, size)
 6.    for location, feature in [detected_locations feature_descriptors] do
 7.       similar_training_features = test_model(model, feature)
 8.       for training_feature in similar_training_features do
 9.          standard_displacements = des-normalize(get_displacements (training_feature),
            angle, size)
10.          predictions = location + standard_displacements
11.          draw(map, prediction)
12.       end for
13.    end for
14.    add(maps, [map size])
15. end for
16. locations, sizes = select_center(maps)
```

descriptors for future retrievement.

In test, the similar feature extraction processing is done for each test image. However, the feature descriptors are extracted from the whole test image. Various scales are used to extract features because we do not know the size of the object beforehand. For each scale, the extracted feature descriptors get their similar training descriptors through the trained model, for each of the recovered descriptors, the associated recovered displacements are scaled up by multiplying the displacements with the scale of the feature descriptor. Then the displacements are rotated according to the query descriptor coordinate system. The scaled and rotated displacements are used to cast votes to predict the center of the object of interest. The prediction location is computed by adding the transformed displacements to the location of the query descriptor. The accumulated predictions generate a vote map in the query image for that scale. For each scale used to extract feature descriptors, a vote map is created. The centers of the object of interest are those location where high number of votes are accumulated.

In the following sections, we present the different parts of the proposed method in more detail.

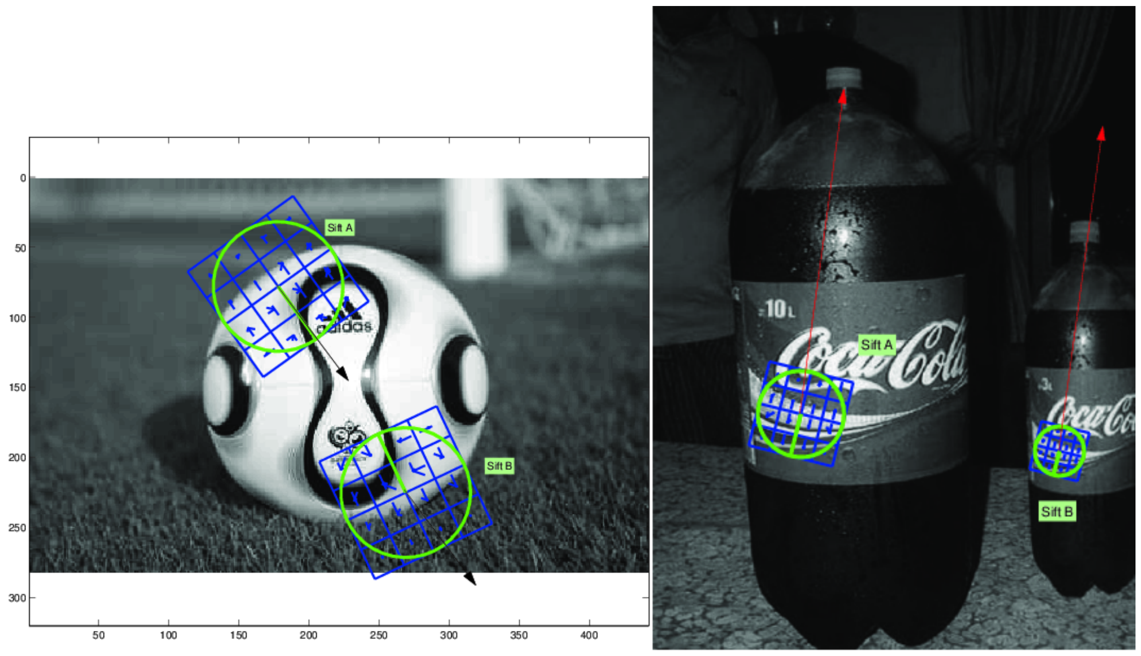


Figure 3.1: Two examples which show the limitations of using displacements without normalization. Left: Two similar SIFT descriptors with different orientation, the recovered displacements for SIFT point B from training data A misses the direction to the center when patch orientation is ignored. Right: Two similar SIFT descriptors with different scales, the SIFT point B failed to get the position of the tap using the displacement directly from the larger feature A.

3.2 Features & Geometric information

In the proposed approach, we take advantage of the geometric information of orientation and scale of the descriptors. The displacements values from each keypoint to the object center are transformed in a new xy-Cartesian coordinate system using the orientation and scaled with its size.

For each SIFT descriptor we keep the relative position information from the SIFT center to the center of the object. Considering that, a similar SIFT descriptor is detected in test, we can use the relative position information saved before to recover the center position of the object from the new SIFT. The scale of the SIFT is also recorded.

However, using the raw displacements mentioned before in the standard x-y axis is not enough when objects are in different orientations. If a feature patch in test has an orientation change, then the displacements utilized to recover the object center could point to another direction. Also, the displacements module may not be precise if a test feature differs in its scale to the scale of the most similar training data. These effects are illustrated in Figure (3.1).

In test, an inverse transformation is performed to the recovered displacements in order to obtain the displacements in the image Cartesian coordinate system. The transformation involves a rotation transformation based on the SIFT orientation and a scale transformation based on the size of the keypoint.

We use SIFT descriptor as our feature descriptor, the reason for using this type of descriptors is because it outputs along with the patch descriptor the orientation and scale of the detection. However, other scale and orientation invariant descriptors could in principle also be used.

3.2.1 Rotation Matrix

A rotation matrix is a matrix which can be used to perform a rotation in Euclidean space. In R^2 the rotation matrix is:

$$Rotation(\theta) = \begin{vmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{vmatrix}$$

This matrix rotates vectors to a new Cartesian system through multiplication:

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$

In this way, displacements labeled in standard Cartesian system can be annotated in new system according to the rotated angle θ. Applying the previously formulation we get:

$$x' = xcos\theta - ysin\theta$$
$$y' = xsin\theta + ycos\theta$$

Note that the rotation applied with the angle θ is counterclockwise. To transform rotated displacements back and recover its values in the standard Cartesian system, the inverse transformation is done by applying the same rotation formulations with $-\theta$.

The effect of this operation does not change the euclidean distance from a point to another, but the displacements x and y with respect to each axis suffers modifications according to the rotation. One example is shown in Figure (3.2) which the coordinate

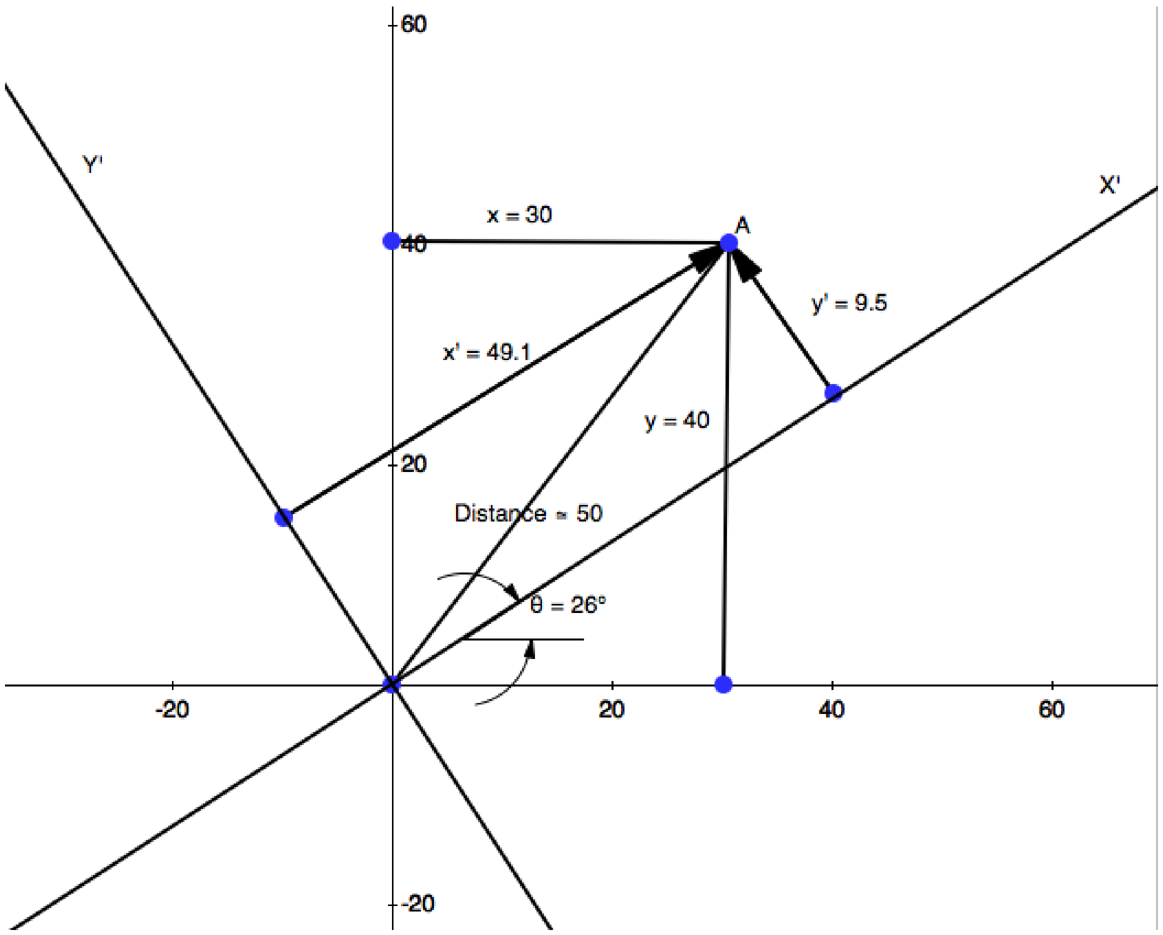


Figure 3.2: A coordinate system rotation example

system is rotated 26 grades, the displacements in the new coordinate system differ to the ones in the standard system. In test, the inverse transformation mentioned before is done to recover the displacements in its original Cartesian system.

3.2.2 Scale normalization

After rotating the displacements as described previously, the new x, y axis is normalized according to the scale of the keypoint. The displacements after orientation normalization x' and y' are divided by the scale of the feature point. In test, to recover the distance from a feature point to the center of a object, the unitary displacements are multiplied by the scale of the test feature point to get the displacements corrected according to the size of the test object. After the orientation normalization, the displacements in the standard Cartesian system is recovered using the process explained in the previous section.

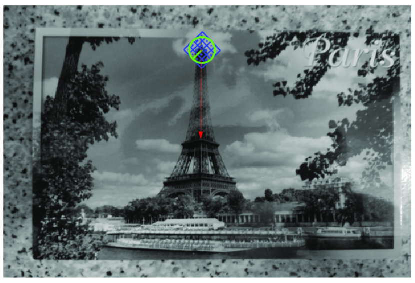
With this normalization process, our recognition system manages to handle scale as well as rotation on the z-axis on the image plane. However, for a multi-scale problem in which each image may contain objects of interest with different sizes, different scales of SIFT patch have to be detected for each image. The best detection of all scales is chosen as the final localization of the object and its scale is then the size of the detected object.

3.2.3 Center voting

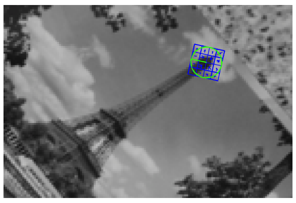
The displacements between each positive SIFT point to the centers of a specific object are normalized and saved for future retrievement. The entire process is described in Figure (3.3 a-h). In this toy example, only one feature point is considered for clarity. The objective of this example is to show how are the displacements normalization and retrievement is done for different orientation and scale changes.

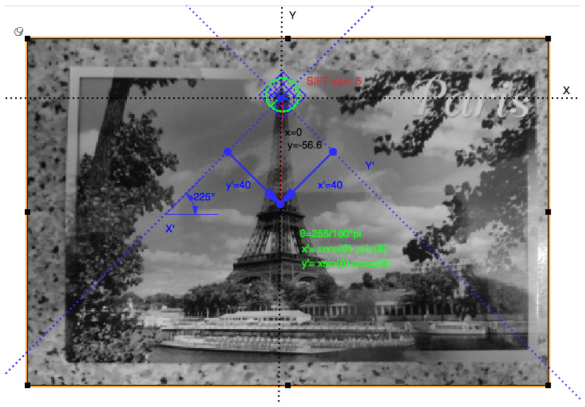
In training, we detect feature points and describe them through a feature descriptor (SIFT in this case). The distances to the center of the object from the detected feature point location in coordinate x-y are recorded. Using a rotation matrix, the displacements in a new Cartesian system which is rotated by the orientation of the SIFT point (3.3d) is obtained. Finally, the displacements are scaled down by dividing by the SIFT scale (3.3f).

In test, for each new image, the features descriptors are extracted along with their sizes and angles. For each feature descriptor the most similar training SIFT descriptors are retrieved (using algorithms like Nearest neighbours or Random Forests) (3.3c). The recovered feature points have the information about their displacements to the center in

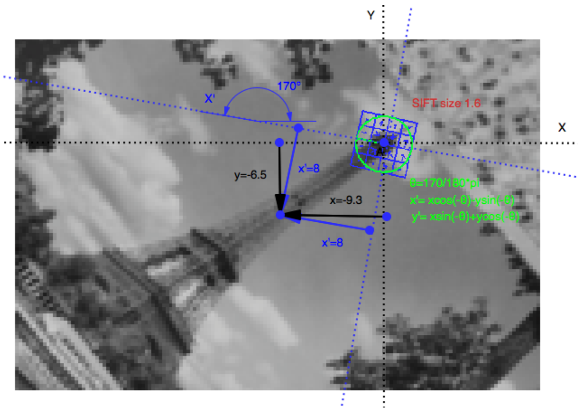

(a) Original training image

(b) Original test image

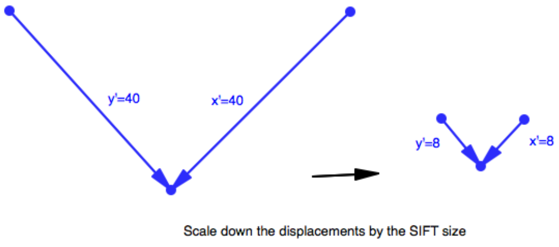

(c) Focused test image



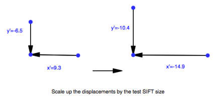
(d) Orientation Normalization



(e) Orientation recovery



(f) Scale Normalization


(g) Scale recovery

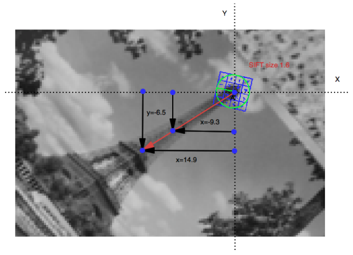

(h) Final prediction

Figure 3.3: One displacement normalization example is shown here. Figures (a,d,f) show a training SIFT descriptor A with its displacements to the center of Eiffel Tower which are rotated and scaled using the SIFT orientation and scale attributes. Figures (b,c,e,g,h) show detections in test with SIFT descriptors similar to point A of the training data. Using the normalized displacements in training and applying the inverse normalization process, we get the prediction to the center of the object.

the coordinate system given by their orientation and scale.The displacement vector and coordinate system of the training keypoint is used to calculate the equivalent displacement vector in the coordinate system of the test point. This is done by rotating back the displacements applying the rotation matrix in inverse direction. The angle used in this case is the negative degree of the orientation of the test feature descriptor, see in figure (3.3e). Finally, the rotated displacements have to be scaled up to get correct displacements according to the scale of the test feature point. This process is shown in figures (3.3g, 3.3h).

The center position prediction of an object is obtained by adding the normalized displacements to the center point location of the test feature. This center voting process is done for each of the image features and a vote map is then created with the accumulated votes in each positions.

3.3 Mesh detection

Most image recognition systems look for keypoints that have high repeatability and stability. The detected points are representative characteristics of the images. In this way, similar images will produce detections in similar locations. This repeatability property is vital for image recognition tasks. However, these detectors might present some disadvantages:

- Few feature points might be detected for the object of interest.
- The numbers of feature points are different for different kind of objects.

For example, using SIFT detector might produce large number of detections concentrated in regions with high contrast. On the other hand, the object of interest might not obtain enough number of detections to describe it, if it has low contrast. In consequence, the objects of interest might not be properly described. This poses some difficulties in the generalization power of the system.

Another way of getting repeatability is to retrieve keypoints from the entire image using mesh detection. Mesh detection method is based on sampling all locations on a grid as shown in Figure (3.4). In this way, a regular net of points are used as keypoint detections. The detected points are then described using any feature descriptor (we used SIFT in our experiment). The reason of using this approach is that, the density of the detected keypoints in an image can be maintained. The mesh spatiality parameter s defines the distances between neighbours vertically and horizontally. With this parameter, the density of keypoints in an image can be easily controlled. The scale of the detection should also be defined beforehand (in our experiments we used the same value s for the spatiality

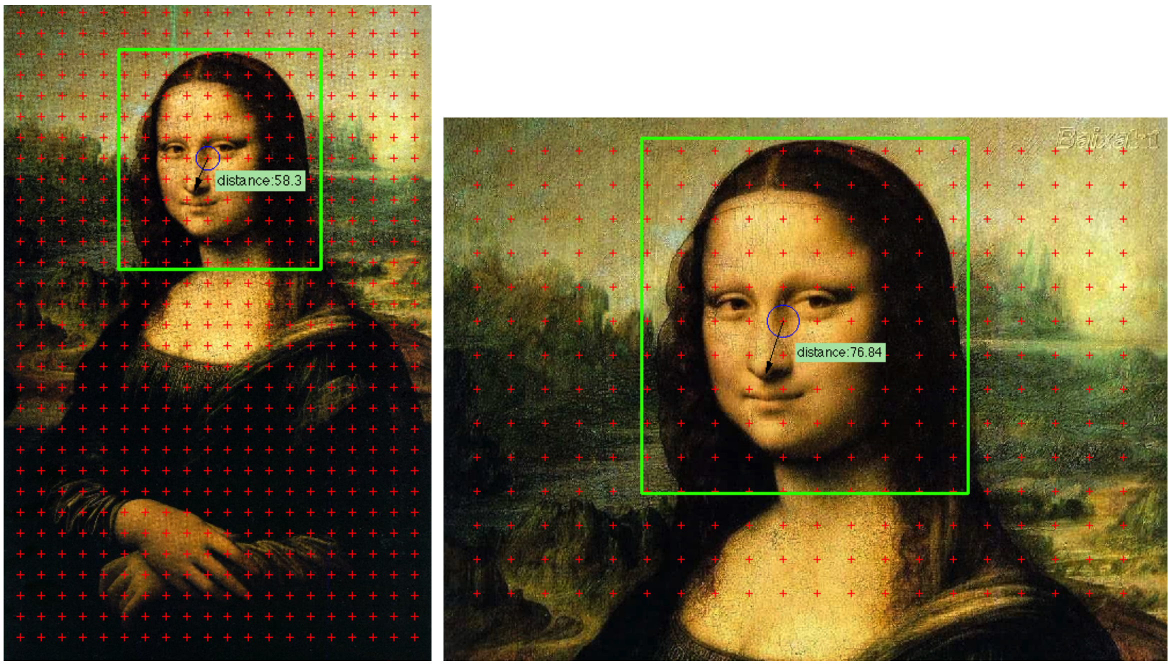


Figure 3.4: The face rectangle in the right figure is 1.3 times bigger than the left one, their sizes of mesh points should be in the same ratio. here the mesh point size and its distance to neighbours a 36, 36 for left figure and 48, 48 respectively for right figure. Note that mesh points are located in similar zones in both images. The distances indicated in the figures are in the same ratio which should be normalized to unitary displacements.

and SIFT scale).

With mesh detection all training objects can be fully covered with descriptors and can get a similar number of descriptors. At the same time background can be ignored. In test, all parts of the images receive the same importance (equal density of detections). In a certain extent, this approach smooths the concentration of out-of-object predictions in a local region with high contrast. On the other hand, positive object regions absorbs more votes in their center which is one of the objectives in a object detection task and the detection performance can be assured.

In our original method, we do not use negative examples (background or out-of-object cases in the image) to train our model, because we think that for a generic object detection problem there could be an extremely large number of negative cases. Using these examples could be of interest when the test images have similar backgrounds. Without negative cases we can create a object detection model which is independent of the scenes where the objects are located. One disadvantage of this approach is that in test we do not know which keypoints are negatives cases, hence all detected keypoints cast votes. This could produce noisy vote maps. This effect can be alleviated by using mesh detection as we explained before.

3.3.1 Determine object size

Once the center of an object is detected, we need to estimate the size of the detected object. For single-scale problem, that is, when the size of the objects in the test set is the same as the size of the objects in train, it is enough to use the size of the objects in train as the size of object.

In multi-scale problem, that is, when the scale of the test objects is unknown beforehand, we can not use the size of the training objects directly to predict the test object size. In this case, mesh detections with different keypoint sizes and different distances between mesh points in test is applied. Various keypoint sizes are tested, and the size that predicts the center position with the highest number of votes is used to determine the size of the object. For this purpose, the ratio between the selected test keypoint size and the size of the training keypoints is calculated. Then the object size is determined by multiplying the size of the object in training by this ratio.

In figure (3.4), an example of this approach is shown, the SIFT point of each example image have different sizes, but they occupy almost same relative region of the face. In this way, we can be able to recover the true displacements for a test keypoint through its most likely training data and use the displacements to make predictions of object center according to the size of the object.

3.4 Training & Similar SIFT retrievement

SIFT descriptors similar to the ones extracted in a test image should be recovered from the set of training SIFTs. Here we have used these different algorithms: K nearest neighbours (KNN) and Random Forests.

KNN is a non-parametric method and it is simple to adapt to our object detection approach. In this method, no explicit training step is required, the model construction in this case consists only of storing the feature vectors and the associated displacements. For each query SIFT feature, the fixed K closest SIFTs with its displacements in training are recovered. However, the nearest neighbours searches are computationally expensive for

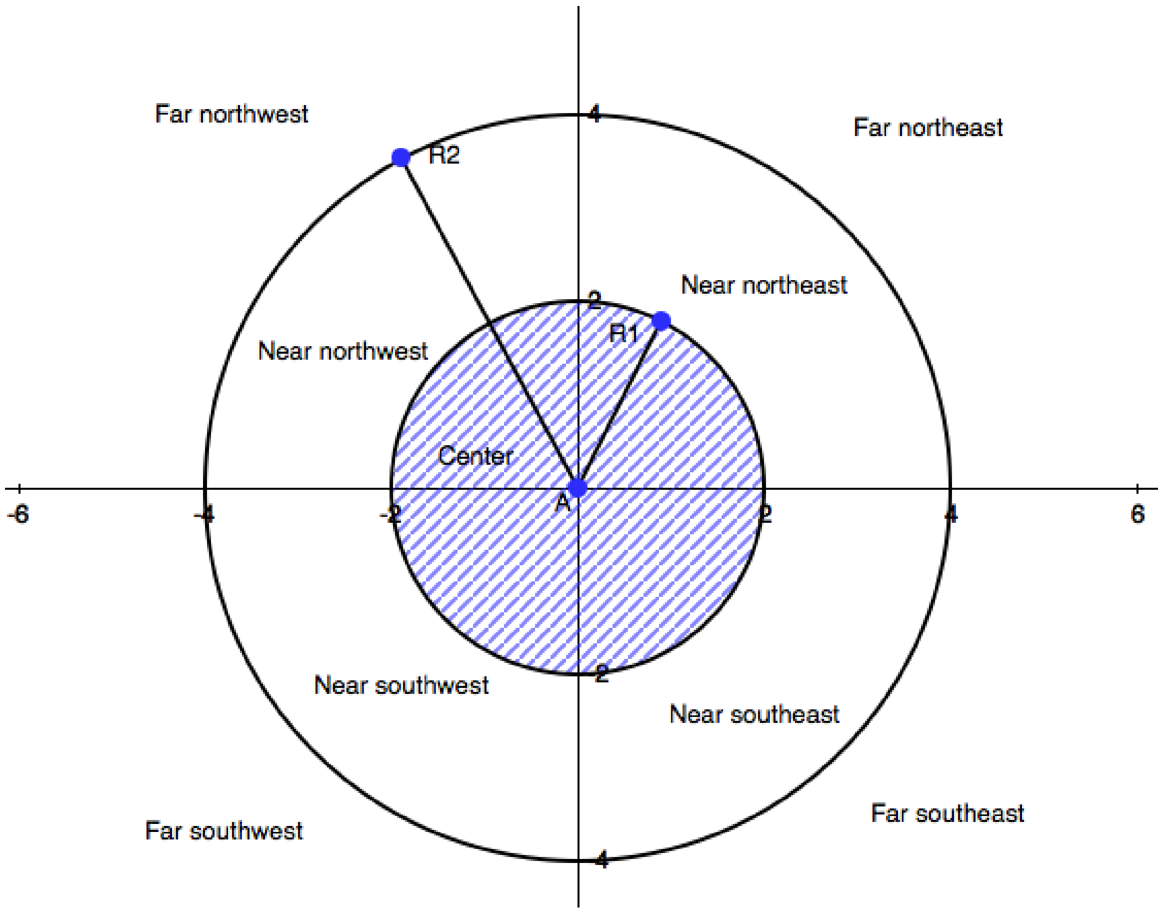


Figure 3.5: The areas of each label used in proposed Random Forests

large corpus of data, data structure such as KD-tree [Bentley (1975b)] could be used to alleviate this problem. The KD-tree is a binary tree in which each non-leaf node can be thought of as a splitting hyperplane that divides the data space in two parts. In this way, the nearest neighbour search could be done efficiently by eliminating large portions of the search space in the beginning. In our experiment, we use this algorithm implemented in the FLANN library [Muja and Lowe (2009)] to perform fast nearest neighbor searches.

Random Forests are trained to learn the displacements from a SIFT descriptor to the center location of the object. This can be done by using positive training descriptors with its recorded displacements to the center of the object. However, in the proposed approach, the displacements are not used directly to train the model. The displacements of each training descriptor are encoded with nine possible classes: center, near northeast, near southeast, near southwest, near northwest, far northeast, far southeast, far southwest and far northwest. In order to label the training examples, two distances parameters have to be defined: the radius R1 which is considered as the boundary between ”near” and ”close”, and the radius R2 which is considered as the boundary between ”near” and ”far”. If the distance between a keypoint to its center is lower than R1, it will be labeled as center. If it is lower than R2, it will be labeled as near. Otherwise, it is labeled as far. For displacements labeled as far or near a further labeling is performed depending on the quadrant they are in by adding NE, NN, SE or SW. Each area with its corresponding label is shown in Figure (3.5) These two parameters R1 and R2 are set in order to have the similar number of training keypoints distributed across all labels.

To train the trees we use the Gini impurity metric to split training examples in each tree node:

$$i(N) = 1 - \sum_{i=1}^{m} f_i^2$$

where f_i^2 is fraction of examples of class i in the set and m is the total number of classes (9 in this case). The split criterion in each node is a binary test: A randomly selected attribute index i and a randomly selected threshold θ compose the binary test: $X(i) \leq \theta$. With this type of binary tests, the data in each node are split into two parts. A number of tests are composed in each node, the one with best Gini impurity measure is then selected to split data in that node. This splitting process continues until the number of examples in a node is smaller then a predefined value or all examples are from the same class

3.5 Method summary

In summary, the training phase (Algorithm (5)) works as following. The image features are extracted from each training image. Positive images are cropped to contain one single object inside. Only features inside the cropped area are kept. And the parameter s (which controls spatiality of the mesh points and it is the size of SIFT feature region) need to be predefined. This parameter is set in order to give around 100-200 detections per object. This gives a reasonable detection sizes in order to capture important features of the object. This stage of extracting descriptor vector includes the following steps:

- Mesh detection to establish the locations of the center of each SIFT feature.
- Describe each SIFT region with a SIFT descriptor. In this step, the orientation of the SIFT descriptor is also returned.
- The displacements from the each mesh keypoint to the center of the training image are calculated.

- The displacements are rotated using the angle given by the orientation of the SIFT region.
- Scale the rotated displacements by dividing the displacements by the SIFT scale s.

Each SIFT along with its displacements are saved in training dataset. This is done for each positive cropped training image. A model is trained using Random Forests or Naive Bayes. For the case of Random Forest, normalized displacement vectors are further encoded into nine displacement categories. This is based on the distance between the detection and the center of the object and the quadrant where the detection is with respect to the center of the object. The radius R1 and R2 have to be set for this propose. The values which balance the label distribution according to all displacements of the training descriptors are selected.

In test, Algorithm (6), the same feature extraction steps are performed for each image, in this case, the extraction is done for the entire image as the object location is unknown. If the size of the object is also unknown, we have to perform mesh detection with different grid and SIFT scales. For each SIFT scale s_i, a voting map is generated by following steps:

- For each image descriptor of scale s_i, their similar training descriptors are retrieved using Random Forests or KNN model. For Random Forests the training descriptor retrieved correspond to the ones stored in the leaves in which the test descriptor is classified. This number can be different for different test descriptor. For KNN, a fixed number of neighbours is obtained for each descriptor.
- For each retrieved training feature, one pair of displacements in x and y directions is obtained for the corresponding test descriptor.
- These displacements for each test feature are accumulated in an image map.

The voting map m_i is generated after applying previous steps for image features of scale s_i. The location of the object is obtained by selecting the position with highest number of votes score for all generated voting maps. Once the location is formed, the scale is determined simply by taking the scale of the corresponding voting map of the location.

Chapter 4

Experiments

In order to analyze the performance of the proposed method for object detection, the following datasets are used: the UIUC Car dataset [Agarwal *et al.* (2004); Agarwal and Roth (2002)] and a moth dataset [Yao *et al.* (2013)]. The detection consists in detecting the location and area of an object in each query image. The UIUC Car dataset is a widely used dataset in object detection research area [Mutch and Lowe (2006); Gall and Lempitsky (2009); Leibe *et al.* (2004)]. The moth dataset is created by [Yao *et al.* (2013)]. This work is a part of a rice pest control system. This dataset will help us to validate the the detection performance of the proposed method with respect to orientation changes (on the z-axis on the image plane) of the objects. To verify the quality of our method, we have tested the same data sets with the state-of-the-art method Class Specific Hough Forests. The datasets are described in the following sections in more detail.

To evaluate the performance of the tested algorithms we used F-measure for each dataset. The F-measure used here is:

$$F = \frac{precision\ +\ recall}{2}$$

This is a variant of the traditional F-measure which is formulated as following:

$$F_{traditional} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

This metric measures the retrieval effectiveness of each method. The Precision-Recall curve is used to evaluate each algorithm. It is generated by changing the decision threshold (In this way, the precision and recall indices are changed too, so as the F-measure). The Precision-Recall curve helps us to view the evolution of the algorithm performance (F-measure score) along with the threshold changes. The pair of precision and recall values which gives best F-measure for an algorithm is denominated F-measure peak. This

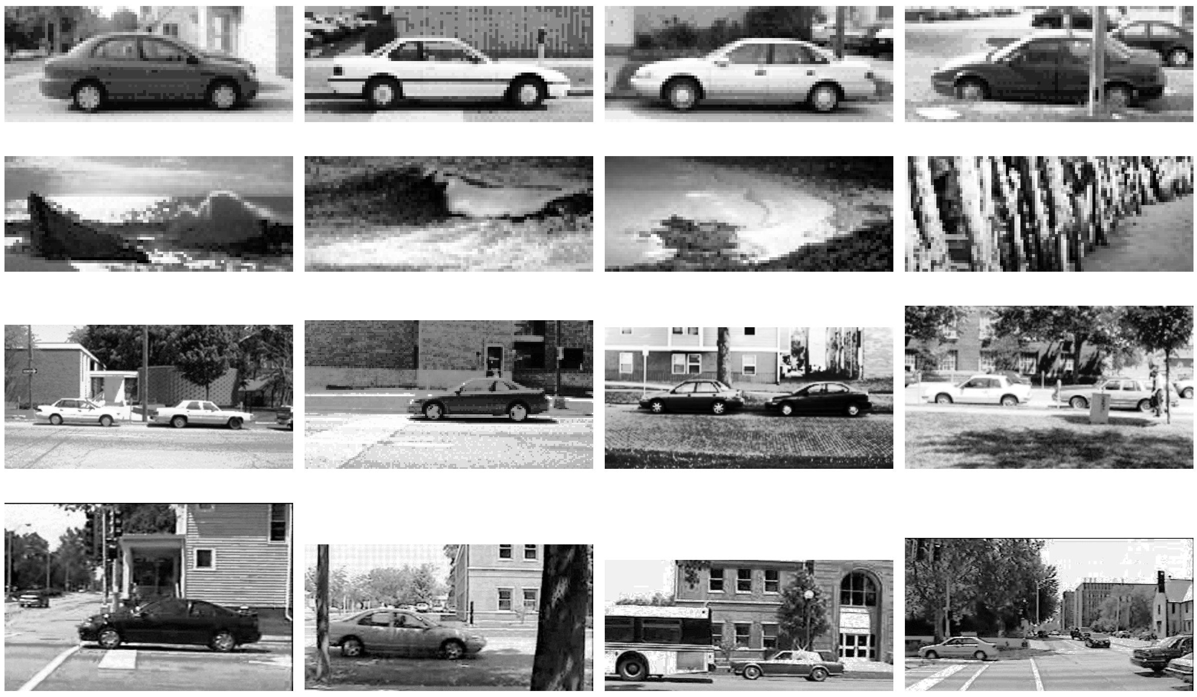


Figure 4.1: Examples of UIUC dataset. Row 1: positive training examples. Row 2: negative training examples. Row 3: test examples for single-scale problem. Row 4: test examples for multi-scale problem with cars in different sizes.

score is shown in the result tables of our experiments and indicates the best performance of each algorithm considering that both false-positive (precision) and false-negative (recall) costs are the same.

4.1 UIUC Car

The images of UIUC cars dataset [Agarwal *et al.* (2004); Agarwal and Roth (2002)] contain side views of cars and the objective is to detect the position the size of each car with a rectangle indicating the region it occupies. The dataset is divided in three parts: A training set containing 550 positive examples and 500 non-car images, a single-scale test dataset with 200 cars in 170 images (with the same size as the cars in the training set) and a multi-scale test dataset with 139 cars at various scales in 108 images. Examples of each part are shown in Figure (4.1).

The positive cases of the train dataset are 100x40 pixels images, each one containing exactly one car that occupies the whole image. The center of each car is then determined as the center of the image. The negative cases are background images. In some of our methods and for the Hough Forests model, these images are utilized as the negative cases to train a model. In other cases, these images are not used.

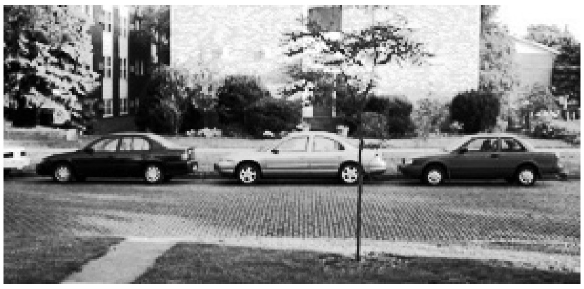
There are two problems that need to be solved in this dataset: a single scale problem and a Multi-scale problem. Both share the same training dataset. In the single scale problem, the cars both in the train dataset and the test dataset have the same size. Hence, only the center of the car has to be detected. In both single scale and multi-scale problems, the test images might contain more than one car. To solve this problem, instead of selecting one maximum from the voting map, we preselect three candidate car locations in an iterative manner per test image. Every time a candidate is selected we draw a rectangle with the estimated object size around the detected center. All pixels included in the rectangle are set to 0, so in the next candidate location selection in that image, none of these positions will be selected as a possible car center location. For each candidate location, we save a tuple which includes the test image index, the predicted locations x, y, the votes response and the estimated car size.

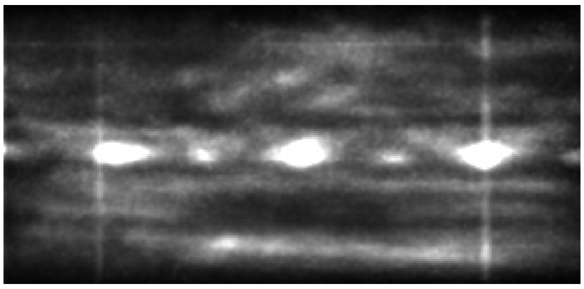
Figure (4.2) shows an example of the car detection process in one single-scale test image. In (a), the original image is shown, the voting map (b) shows the regions that concentrate the highest number of votes. In (c,d,e), the car detections are performed iteratively, each time a car is detected, its region in the voting map is set to 0. In (f), all detections are enclosed by rectangles.

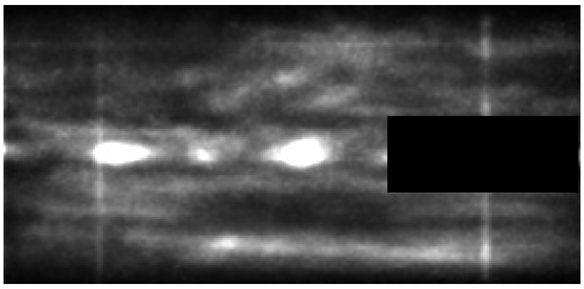
A total of 510 (3*170 images) locations are selected from single-scale test dataset. Once all the predicted centers are selected from all test images, we create a ranking with locations from highest vote response to low vote response. Finally, the first n highest response locations are used to calculate the F-measure with the program provided by UIUC Car dataset and a Precision-Recall plot is generated. The Precision-Recall curves are calculated by varying the size n from 0 to the number of the true cars in the test dataset.

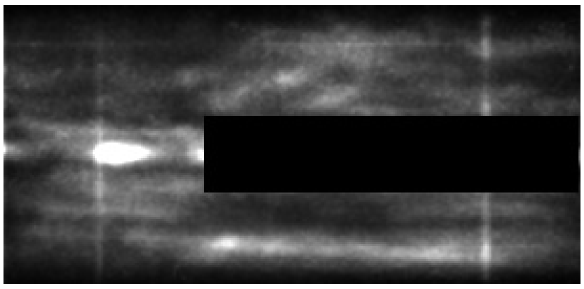
In the multi-scale test set, cars may have different sizes. In order to be able to detect all possible car sizes, detections at different scales are computed. Each scale will give a different vote response maps. The used test scales have to include all possible car sizes. For the mesh detection approach, the parameter s which controls the spatiality of the detections is set to 13 different values ranged between 6 pixels and 18 pixels. For the Hough Forests approach, we use 13 scales ranging uniformly from 0.8 to 2 times the size of the image. For each of these algorithms, 13 voting maps are built.

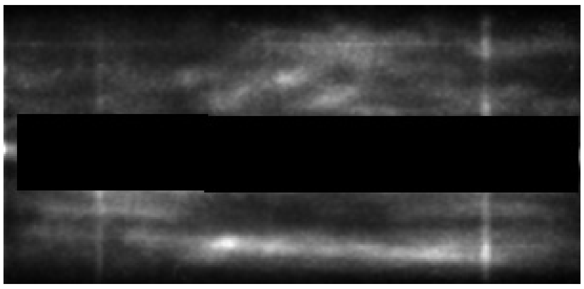
For each of the generated 13 voting maps, two locations where the response in the vote map are maxima are selected as possible car centers producing a total of


(a) Original test image


(b) Before the detection


(c) First detection


(d) Second detection


(e) First detection

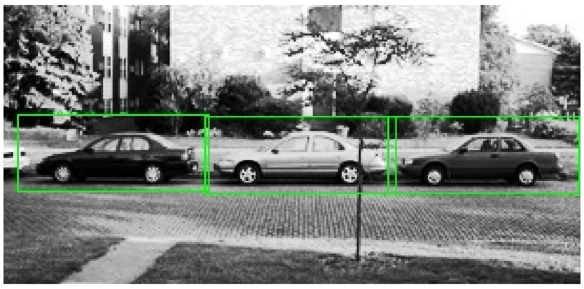

(f) Final results

Figure 4.2: An example of extraction of cars in single-scale problem. The voting map are generated from Random Forests. Once a car is detected, the pixels in the region that are believed to contain the detected car are set to 0.

26 possible locations for each test image. The 26 preselected candidates are ranked according to the vote response from highest to lowest. For each preselected candidate, a rectangle is drawn around it. The size of the rectangle is set according to the scale used to generate the vote map. Then the two most voted positions are selected with the condition that the second one is not in the rectangle of the first center position. Finally, we rank all 216 (2*108 images) selected positions for the whole test set with respect to their votes response. Similarly, as in the single-scale case, we select the n highest response locations to be used to calculate the F-measure variant proposed before and a Precision-Recall curve is than generated varying the number n of the selected car regions.

Different methods have been tested with this dataset for both the single and the multi-scale problem. The following list indicates the methods tested. The details and parameters used for each algorithm will be explained in the following sections.

- Class Specific Hough Forests + Single-scale problem (HF-S)
- Class Specific Hough Forests + Multi-scale problem (HF-MS)
- Mesh-KNN + Single-scale problem (MK-S)
- Mesh-KNN + Multi-scale problem (MK-MS)
- Mesh-Random Forests + Multi-scale problem (MRF-S)
- Mesh-Random Forests + Multi-scale problem (MRF-MS)
- Sift-KNN + Single-scale problem (SK-S)
- Sift-KNN + Multi-scale problem (SK-MS)
- Sift-Random Forests + Single-scale problem (SRF-S)

The combination of Sift-Random Forests + Multi-scale problem (SRF-MS) is not included in the experiments due to its low performance.

4.1.1 Class Specific Hough Forests

The Class Specific Hough Forests algorithm [Gall and Lempitsky (2009); Gall *et al.* (2011)] used here as the benchmark algorithm. It is trained using both positive and negative training examples. Ten trees are built both in single-scale problem and multi-scale problem. Each patch is based on HoG values and have a fixed size of 16x16 pixels. In [Gall and Lempitsky (2009)], a 98.5% F-measure peak is achieved for the single-scale problem and a 98.6% for the multi-scale problem. Although the authors have not reported their exact experiment configurations, like the number of the trees used or the number of scales used for multi-scale problem, we have obtained similar results with respect to the ones reported in [Gall and Lempitsky (2009)].

4.1.2 Mesh detection approach

In the mesh detection approach, as we explained previously, we do not use negative examples to train the model, only the 550 positive images are used for training. A single model is trained for both the single scale and the multi-scale problems. The keypoints are drawn uniformly and equidistantly. In training, the distance from each keypoint to their neighbour keypoints is set to four pixels. All keypoints are extracted with a size of eight pixels. The keypoints are described using SIFT descriptor. Note that the cars in this problem are side views and no variation in orientation exists.

In test, the same configuration as in training is used for the single-scale problem. For the multi-scale problem, various scales of keypoints are used to perform the mesh detections. For the mesh detection approach, 13 grid scales s are used to detect cars with different scales. The values used range from $s = 6pixels$ to $s = 18pixels$. The separation of consecutive neighbour detections is $\frac{s}{2}$. The reason of maintaining this relation between the scale of keypoints and the distance between neighbours detections is to guarantee a equivalent density of detections for each test scale in an each image.

Two classification algorithms are used with mesh detector: Random Forests and KNN. The parameters used in these models are:

- Random Forests: 50 trees are built both in single-scale and multi-scale problem. Class label is based on the relative distance from each SIFT keypoint to their center explained before. In our experiment, R1 and R2 are set to 1 and 3 respectively after testing various times different values. The splitting process is continued until, in each leaf node, there are less than 10 examples or when all examples are of the same class.

- KNN: This algorithm does not need a training process. However, the training data have been structured using FLANN library [Muja and Lowe (2009)] in order to increase the speed of nearest neighbour searches. We also tested KD-tree but for high-dimensional space problems such as the 128 dimensions of SIFT points, this algorithm is slow in retrieving similar data. Finally, we use FLANN library because it provides an automatically selection of data structure and algorithm to speed up KNN searches according to the problem dimension. The unique parameter needed to be set here is the number of neighbours k picked for every test SIFT point. We set it to 20. Other values for k were tested, the differences in the final accuracy of the system for values of neighbours from 5-50 were small.

4.1.3 SIFT approach

In addition to computing a grid of keypoints, the SIFT detector algorithm was also tested. With this approach, we only extract SIFT keypoints whose locations are given by the maxima and minima of the result of difference of Gaussians functions applied in scale space [Lowe (1999)]. Only positive cases are used in the training process. In theory, with SIFT detector low contrast candidate points will be discarded in the detection, that would ensure that the keypoints are more stable and repeatable. However, as mentioned previously, this might cause concentration of detections in background regions which is not desirable for objects detection.

The configuration of the Random Forests for this case is the same as in the Mesh approach and the number of nearest neighbours selected for each test case is set to 5 instead of 20. This is because in this case, we have much less SIFT descriptors.

In order to try to improve the performance of the system when SIFT detector is used, negative example images were also included into the training dataset. We believe that there are many background SIFT points in the test images, and predictions from those keypoints confuse the real locations of the center of a car. In order to reduce the influence of background detections in test images we included background detections in training with the following configurations:

- SIFT+KNN: The base method.
- SIFT+KNN+Negative cases: We adapt negative cases in our algorithm, the algorithm ignores votes from negative nearest neighbours
- Pyramid+SIFT+KNN: Detecting SIFT keypoints with ”pyramid” parameter in detector function of OPENCV which detects keypoint in multiple scales. More detections are made.
- Pyramid+SIFT+KNN+Negative cases: In this case, we combine all previous methods.
- Pyramid+SIFT+KNN+Negative cases+Weighted votes: In this case, we combine all previous methods and use weighted votes. For each test case, we count the positive examples from its nearest neighbours and the weight of each positive example’s

vote is calculated as: $\frac{positive\ cases}{Total\ nearest\ neighbours}$. The idea is to give more importance to detections where all their neighbours come from positive examples.

For the multi-scale problem with SIFT approach, we used the best performance method for single-scale problem, this is the combination Pyramid-SIFT+KNN with Negative cases+Weighted votes. Once the center of a car is detected we have to verify the size of the car. In the mesh approach, the size of the object is directly obtained from the size of SIFT detections, because all keypoints have the same size. However, with in this approach, the keypoint size is different for each detection. For this propose we record the SIFT keypoint size of both training and test data. For each vote, we calculate the ratio of sizes between a test example and its corresponding similar training data. Therefore the scale of a detected car is the mean of ratios between the sizes from test and training data. We only use these ratios of the votes whose weight is above 0.8.

4.1.4 Results

Table (4.1) shows the result obtained in our experiments. The first two rows indicate the obtained result of Hough Forests for both single and multi-scale problems. We used this algorithm as our benchmark algorithm. The four following rows present the results using Mesh detection approach. These methods achieve performances similar to Hough Forests. With the mesh detection approach, KNN performs better than Random Forests. The last rows show the results for methods using SIFT detector. These results are very poor compared to the other approaches.

Table 4.1: Performance at F-measure peak for the UIUC car dataset

Methods + Problem type	F-measure
Hough Forests + Single-scale problem (HF-S)	99%
Hough Forests + Multi-scale problem (HF-MS)	98.58%
Mesh-Random Forests + Single-scale problem (MRF-S)	96.5%
Mesh-Random Forests + Multi-scale problem (MRF-MS)	94.4%
Mesh-KNN + Single-scale problem (MK-S)	99.5%
Mesh-KNN + Multi-scale problem (MK-MS)	99.28%
Sift-KNN + Single-scale problem (SK-S)	82.05%
Sift-KNN + Multi-scale problem (SK-MS)	79.1%
Sift-Random Forests + Single-scale problem (SRF-S)	40.1%

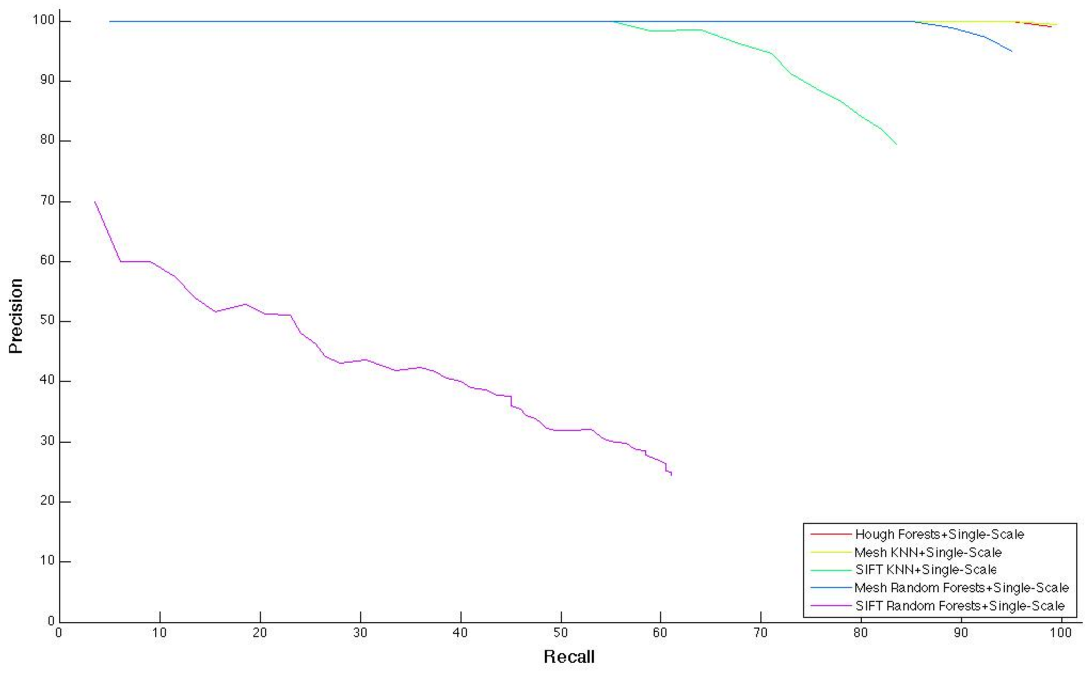
In figure (4.3), the Precision-recall curves of all the tested algorithms for both single and multi-scale problems are shown. The x-axis measures the recall of each algorithm and the y-axis measures the precision. Figure (4.3(a)) shows the Precision-Recall curves for the single-scale problem. Figure (4.3(b)) shows the Precision-Recall curves for the multi-scale problem. The curves of algorithms with good performances are on the top of the figures and decrease sharply when the recall is near to 100%. This is due to the fact that all detected true-positives are selected and large amount of false positives are accumulated in lower positions of the rank.

With Hough forests, we have obtained better results with respect to the ones presented in the original paper for the single-scale problem: 99% of F-measure peak is obtained in our experiment and 98.5% of F-measure peak is reported in their work. In the multi-scale case, each image is scaled multiple times in order to find cars of different sizes. A value of F-measure peak of a 92.47% is obtained when using 13 scales uniformly distributed between 0.8 and 2 times the images size. This value is far from the reported result in [Gall and Lempitsky (2009)]. We believed that if we add more scales to transform each image might increase their performance. A second experiment with 26 scales ranged uniformly between 0.8 to 2 is carried out. With this configuration, 98.56% of F-measure peak is obtained.

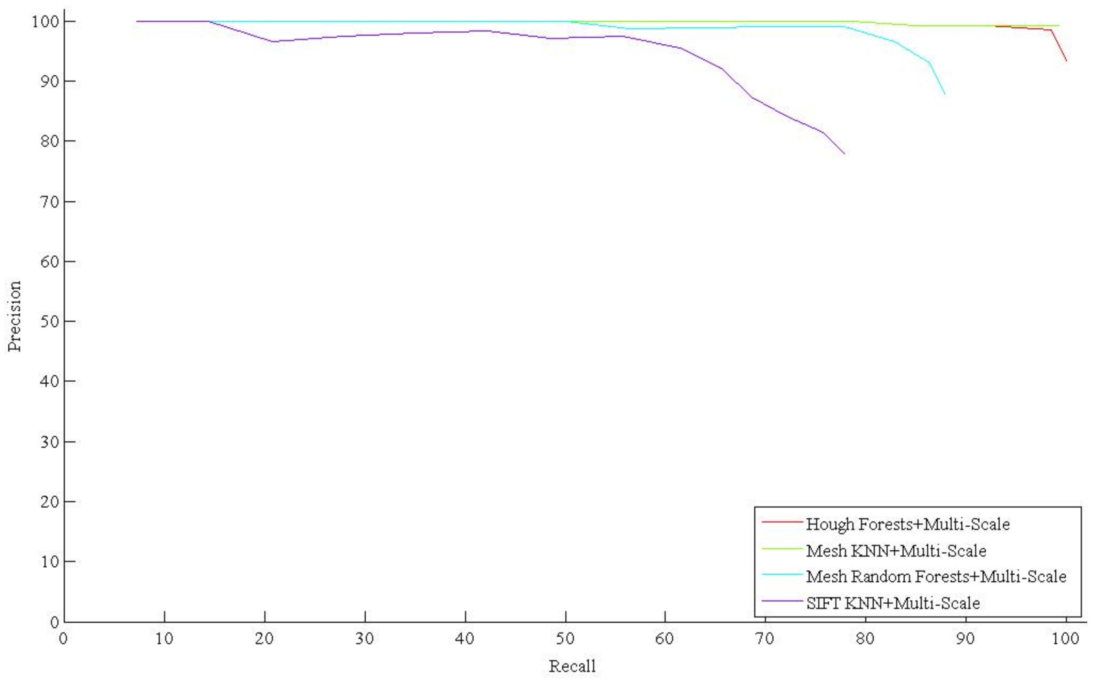
As the results shown in table (4.1), KNN approach outperforms the rest of the algorithms. This combination of mesh detection approach and KNN outperforms Hough Forests, one of the state-of-the-art algorithm. In the proposed algorithm, 13 different mesh scales are used to select object size instead of 26 scales used in the Hough Forests. A drawback that KNN presents is that it is the slowest of the tested algorithms in spite of using accelerating algorithms such as the ones included in FLANN. On the other hand, Random Forests does not perform as well as KNN and Hough Forests, but it is very fast getting similar neighbouring training descriptors for each test SIFT descriptor.

The results for both Random Forests and KNN in single-scale problem indicate that the performances of the system using the SIFT detection approach are very low: 70.25% and 40.1% respectively. In figure (4.4), we can see the difference of the generated voting maps generated of this method and the Mesh method using KNN. From the figure (4.4) we can observe that the position of the center position is much more clear when mesh detection is applied. Despite the poor performance of the SIFT detection approach, we carried out further experiments trying to compensate for its low performance with other mechanisms in conjunction with KNN, such as: using negative examples, pyramid sift detections, weighted votes. The F-measure of each combination of methods is shown in table (4.2) and the Precision-Recall curves are shown in figure (4.5). The results are still far from the best approach using mesh detections. We can see that the number of SIFT points influences substantially the performance of the algorithm.

Some error examples for the different methods are shown in figure (4.6). Most of



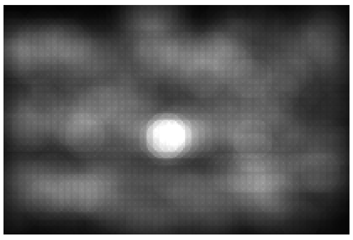
(a) Precision-Recall of experimented algorithms for single-scale problem



(b) Precision-Recall of experimented algorithms for multi-scale problem

Figure 4.3: Precision-Recall curves for all UIUC Car experiments

(a) Original test image with a car in the middle


(b) Voting map using Mesh approach

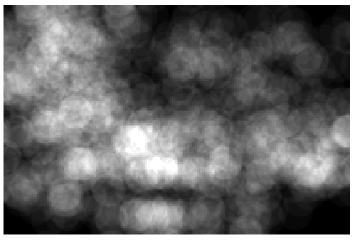

(c) Voting map using SIFT approach

Figure 4.4: Examples of voting map using different methods

Table 4.2: Performance at F-measure peak for single-scale problem for SIFT approach in UIUC car dataset

Methods	F-measure
SIFT+KNN	70.25%
SIFT+KNN+Negative cases	74.5%
Pyramid SIFT+KNN	76.84%
Pyramid SIFT+KNN+Negative cases	81.57%
Pyramid SIFT+KNN+Negative cases+Weighted votes	82.05%

these error are due to the low vote peak of detections, see figures (4.6(a-f)). In multi-scale problem, the deviation in the scale estimation also causes errors in some cases, see figures (4.6(g-i)). Analyzing the results we realized a true positive in multi-scale problem was miss-labeled in the original dataset, see figure (4.7). What we do is manually add it as a positive case in the dataset and all multi-scale Precision-Recall curves are tested using this new version of true car position file. In conclusion, we have tested various methods for UIUC CAR problem, the new proposed Mesh-KNN algorithm outperforms other tested algorithms including the benchmark algorithm: Hough forests. According to the results, KNN based algorithms outperforms Random forests based algorithms. As we proposed previously, mesh detection effectively diminishes the importance of votes from background region in contrast of SIFT based detection and achieves much better performances.

4.2 Moths

Image object detection can be used also in biomonitoring to detect different species. For example in [Yao *et al.* (2013)], the authors use segmentation techniques to separate rice pests and monitor the population dynamics of rice pests in paddy fields. The generated statistics could be used by scientists to control the pests. In our experiment, we have used

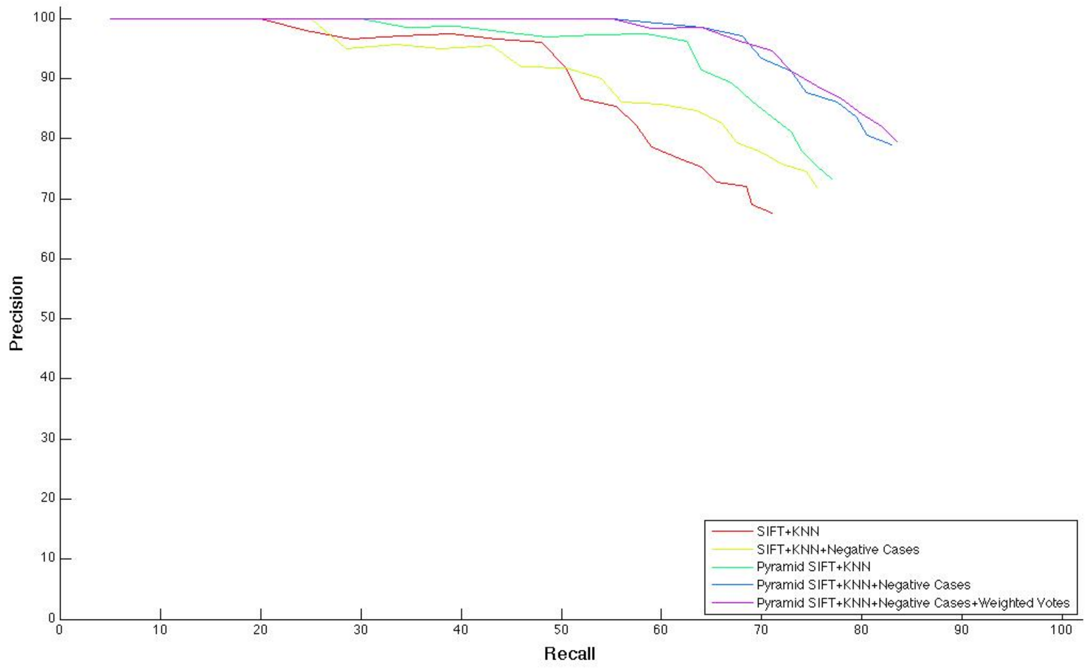
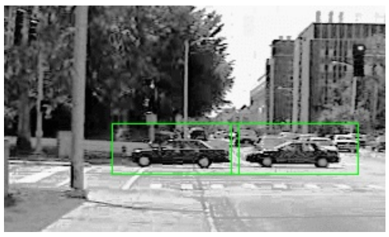
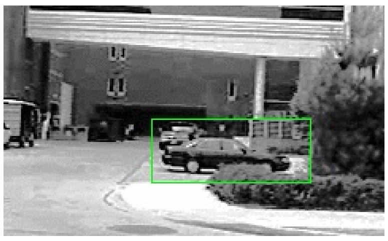
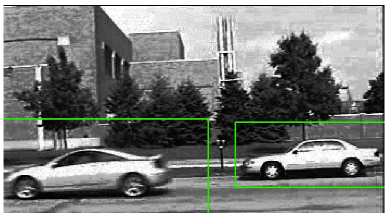
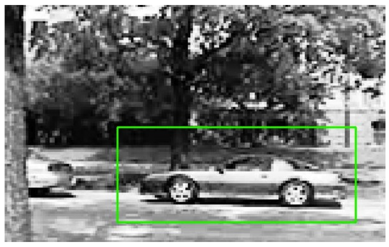


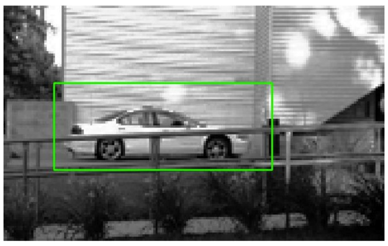
Figure 4.5: Precision-Recall of experimented SIFT+KNN variants for single-scale problem

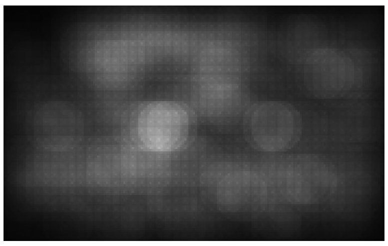

(a) HF-S

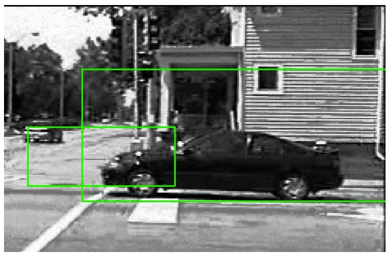

(b) HF-S

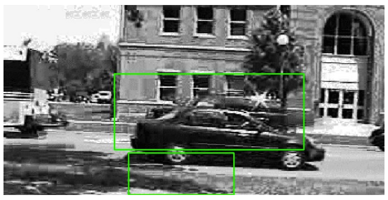

(c) MK-MS

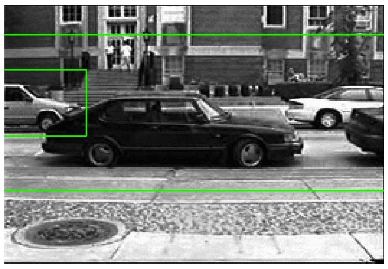

(d) MRF-S

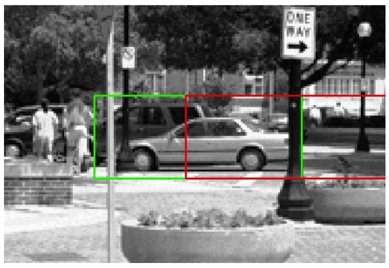

(e) MK-S


(f) Vote response map

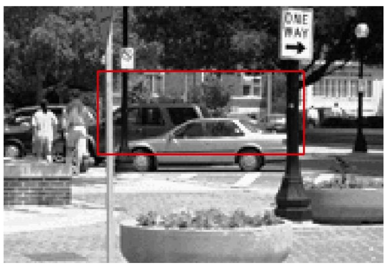

(g) MRF-MS


(h) HF-MS


(i) HF-MS



(j) MK-S



(k) HF-S

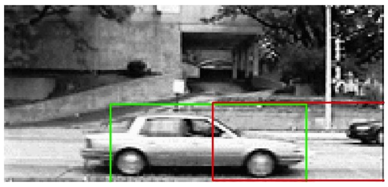

(l) MRF-MS

Figure 4.6: Some error examples found for the different algorithms in Figure (a-i). We can see in these examples that some cars are detected[a-f], but their center positions give a low vote response. Therefore, after all center positions are sorted, these predictions are not included in the final decisions. Figure (f) is the vote response map of figure (e)-the unique failed case in MK-S, in this case the background of the image has a similar color to the car. In other cases (g-i), the scale estimations failed. Figures (j-l) are some false positives (Shown in red rectangles) that obtain high positions in final rank.

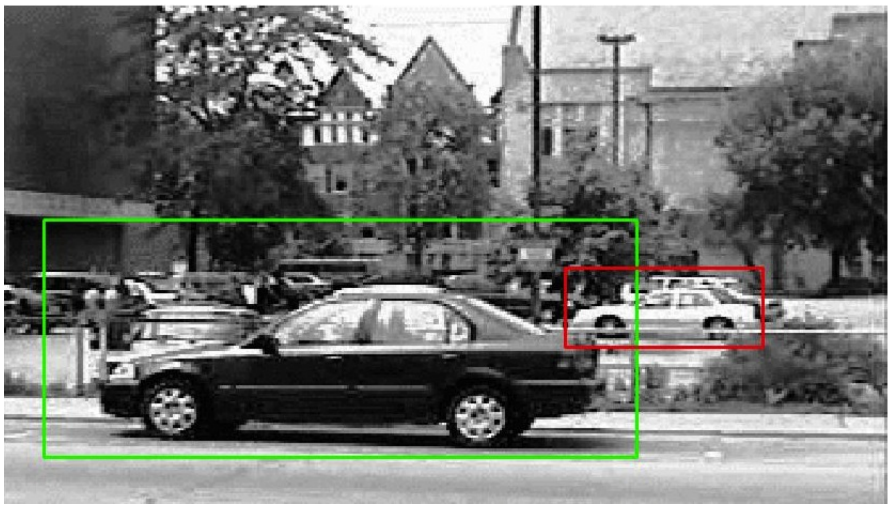


Figure 4.7: A true positive located by our system (in the red rectangle) which is not labeled in the original test dataset.

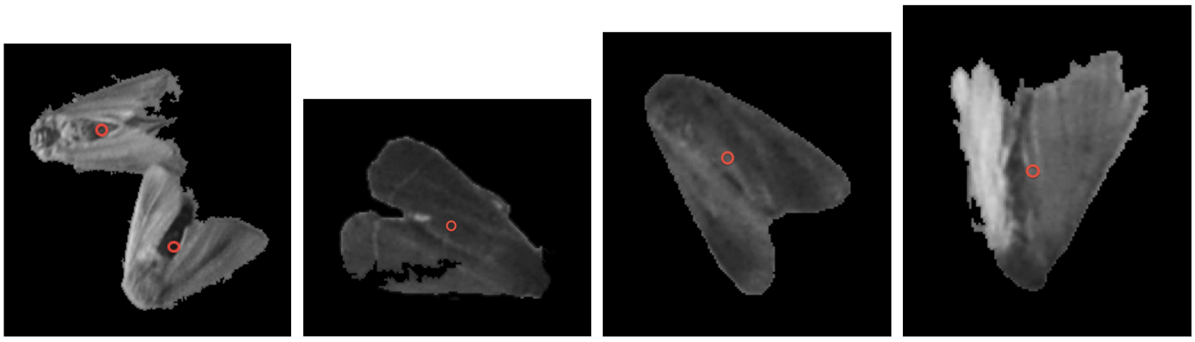


Figure 4.8: Some of extracted moths from the training image. The center of each moth is indicated with a red circle.

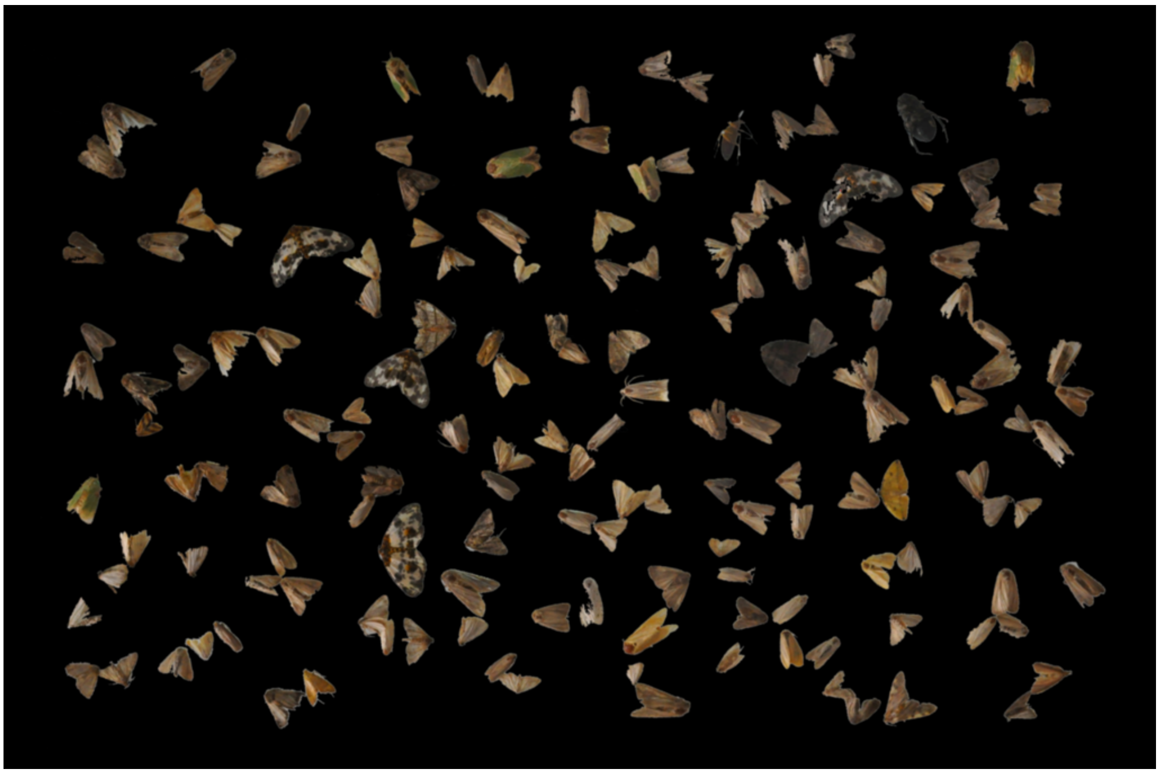


Figure 4.9: A image which contains moths. In training, only those moths which are not touched to others moths are used.

object detection techniques to detect centers of moth which may help us also to determine moth classes. The center detection can be used as a moth counting application to count the number of moths contained in an image. This application presents some challenges such as detecting moths that are touching, see for example figure (4.8.a).

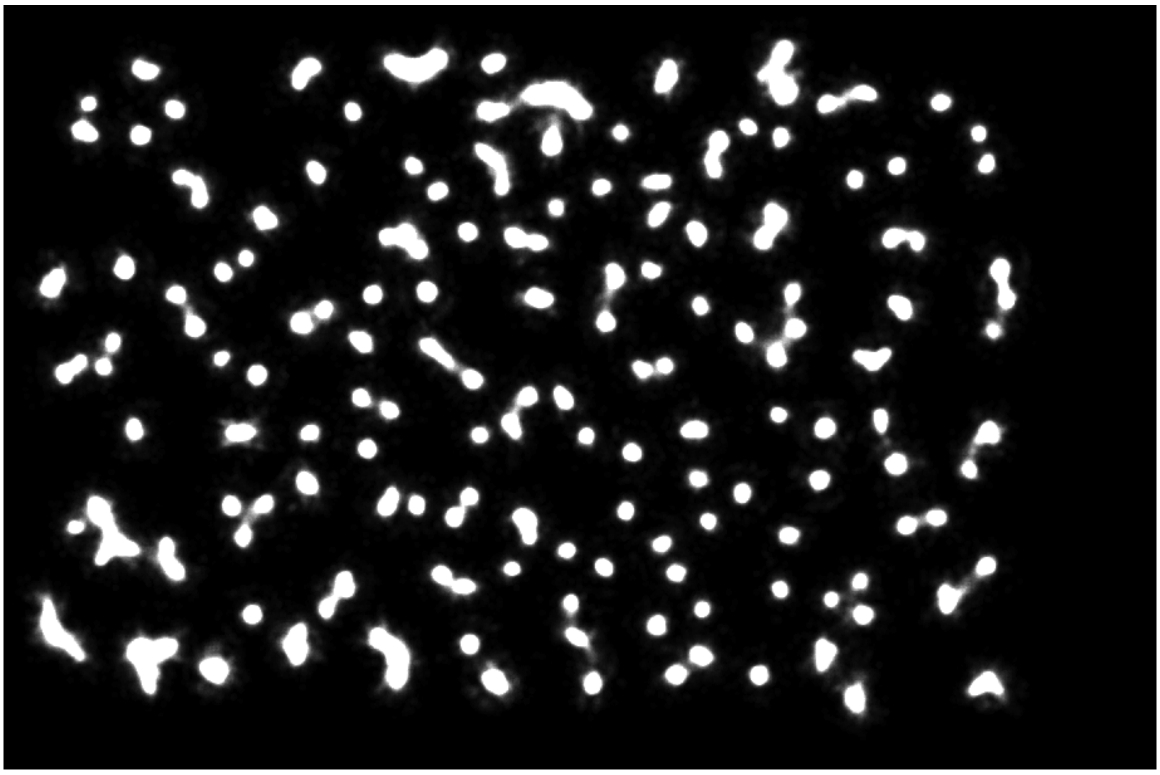
Figure (4.9) shows the training image used. This image has the background removed automatically to simplify the moth detection task. However, there are distortions caused by the background removing process. Some moths may have irregular contours as shown in figure (4.8.b). In our experiment, a manual moth preselection is done to build the training set. We selected only isolated moths that are moths that do not touch any other moth in the image. The training dataset used here is an image which contains about 82 single moths. Using only isolated moths in training will produce more stable center predictions in test. This is because these moth samples are much more homogeneous and are less noisy data. This property will help with the generalization capability of the system. On the other hand, touched moths have the reverse properties, learning the statistics of all the ways in which moths could be touching is almost impossible. Including them in training may deteriorate the performance of the object detection system. Hence we preferred to reduce the training dataset by excluding the touching specimens than to have a large dataset with more complex statistics to learn. In order to extract each training sample, first we transform the entire training image into connected components separated with black background. After that, we select manually each training moth if the connected component contains only one moth. Then the center of each moth is calculated automatically as the center of mass of the connected component. In figure (4.8.b-4.8.d), some gray-scaled training examples are shown. In this experiment, we do not use negative examples to train our model.

The test dataset consists in another image like Figure (4.9) that contains different specimens. The objective is to detect all moths in the image including the ones that are touching. We have annotated manually all moth centers in the image for the performance evaluation. The idea is to detect all moth centers. Note that each moth is in different orientation and has different size and a priori we do not know its orientation and size.

In test, SIFT descriptors are extracted in a grid. For each image feature descriptor the most similar neighbours in training data are retrieved. For each retrieved training point a prediction is made. The predictions are displayed in a vote map, see figure (4.10a). After the voting map is generated, maxima are ranked from higher to lower. In this problem, we do not know the size and the shape of each test moth, so we can not change the moth region to 0 directly in the voting map once the center is selected (to prevent future selections in this area) as we did in UIUC car. It is necessary to eliminate the moth region to prevent future center detections to fall in the same region which would be inconsistent.

Two approaches have been used here to get moths centers: local maxima and back-projection:

- Local maxima: The voting map returned from previous module has been applied a smooth filter twice to remove possible noise. The filter is a disk convolution filter with a radius 10. An example is shown in figure (4.10). In figure (4.10a), the vote map is drawn in gray-scale where the pixels are ranged from 0 (total absence, black) to 1 (total presence, white), the regions with high number of votes are in white (Saturated in this example). In figure (4.10b), a smoothed map is presented, this map is drawn in the full range of its values, those orange pixels indicate positions with a high number of votes. Once the vote response map is generated and refined, the center of each moth has to be recognized. First, we divided the image in different connected components using the black background information. However, each connected component may have more than one moth in it, all maxima in each component's region from the voting map are considered as the possible centers of the moths and a ranking is created in a descending order according to the number of votes in each maximum.


(a) vote map

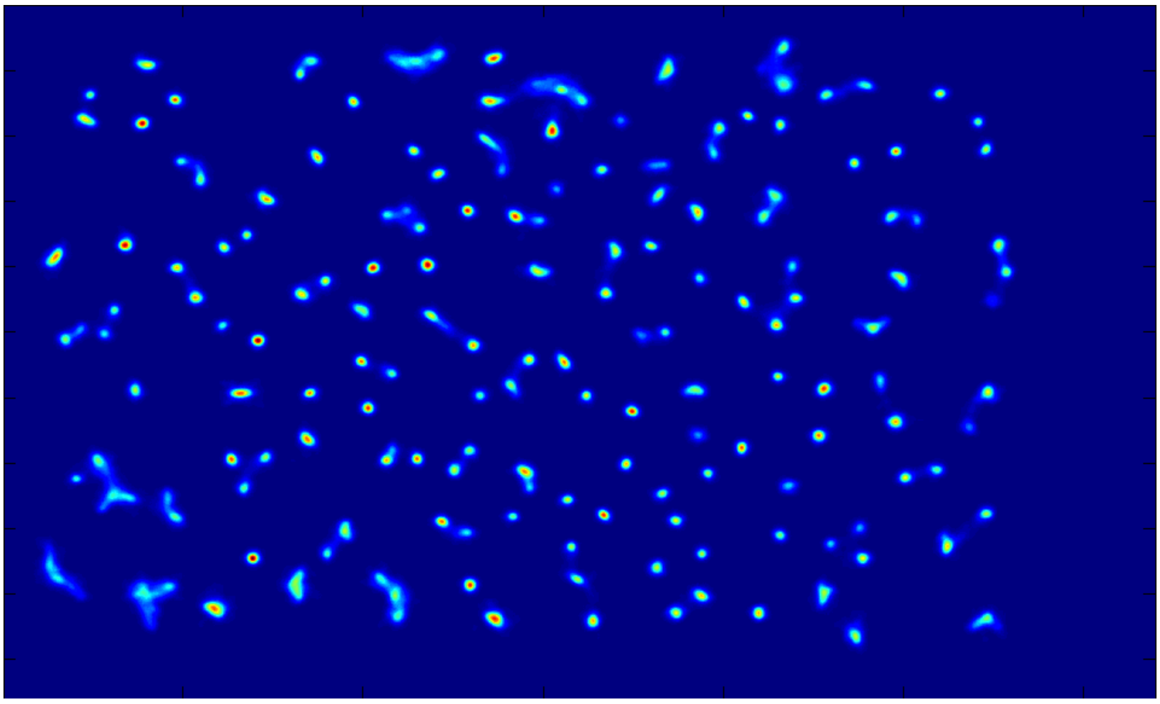

(b) smoothed vote map

Figure 4.10: Example of a vote map with its smoothed version, the center positions are more clear after applying the smooth operation. The maxima of each spot are selected as the moth centers.

- Back-projection: In this method, each test example finds its most similar points in training data using whatever classification method, each of these examples cast one vote. In the vote response map construction procedure of this type of method, we save the origin position (location of the test example) and the predicted position for each vote. Once a maximum is selected, we remove this center region (rectangle of 11x11 with the selected center position in the middle) and set all pixels in this region to 0. We also lookup for the voters V which have predictions in this region. Then we remove all others predictions from these voters V in the voting map. All selected maxima are ranked in a descending order as the center predictions like in the Local maxima approach.

The selected maxima are used to measure the performances of the algorithms. As we do in UIUC Car problem, we selected first n predictions to calculate the F-measure and we vary the n to generate a Precision-Recall curve. Each connected component from the test image may have a number C of real moth centers and a different number P of center predictions from the selected n predictions. To decide if one prediction is correct or not, we have used the best-first algorithm (7). First, the distance of each pair of center prediction and expected center position is calculated. The distances are sorted in ascending order and stored in a list with its expected center and predicted center indexes. Then we pop out from the list each tuple. In our test configuration, we considered that a predicted position is correct only if the predicted center is less than 15 pixels away from the real center position. So if the distance is less than 15 and the expected center and predicted center indexes have not been checked previously, a true positive is added. The number of false positives is the difference between the total number of predicted centers and the true positives. The number of false negatives is the difference between the total number of expected centers and the true positives.

4.2.1 Beam Angle

In addition to SIFT descriptors, for this problem we tested also beam angle descriptor [Arica and Vural (2003)]. For the moth dataset we can easily extract the boundary of each connected components as the images have uniform black background.

The selection of variable K of the K-curvature in this algorithm is important. Small K means that few neighbours points around the salient point are used. The more neighbours points are included in the histogram creation, the more global the boundary information that is included in the descriptors. In our experiment, we use three different values: 50%, 25% and 12.5% for the percentage of points of the boundary around the salient point to be included. These sub-histograms are normalized once are created. However,

Algorithm 7 Best-first decision verifications

```
$C \leftarrow expected\,centers$
$P \leftarrow predicted\,centers$
**for** $y_i$ in $C$ **do**
    **for** $x_i$ to $P$ **do**
        Calculate the distance between $y_i$ and $x_j$
        Save it into a priority queue $Q$ which contains three fields:
        Distance, $y_i$ and $x_i$
    **end for**
**end for**
sort($Q$) by distance from the lowest to higher.
set $list_x$ empty
set $list_y$ empty
true_positives = 0
**while not** empty($Q$) **do**
    distance, $y_i$, $x_i \leftarrow$ pop_top($Q$)
    **if** distance $<$ 15 **and** $y_i$ **not** in $list_y$ **and** $x_i$ **not** in $list_x$: **then**
        add($list_y$, $y_i$)
        add($list_x$, $x_i$)
        true_positives = true_positives + 1
    **else if** distance $>$ 15 **then**
        break
    **end if**
**end while**
false_positives = $length(P)$ - true_positives
false_negatives = $length(C)$ - true_positives
```

each sub-histogram have a different weight according to the percentage of points in the sub-histogram: The more sampled points, more weight the histogram receives. So the values of these three sub-histogram are multiplied by 800, 400 and 200 respectively. Then these weighted sub-histograms are appended and the final histogram is created. Aside from the histogram, we record also the tangent of the salient point in the curve and the number of the points in the boundary. The distance from a descriptor of salient point in a boundary to the center of the object is rotated by the angle formed by the tangent and the x-axis.

After generating all Beam angle descriptors, we used KNN and Random Forests for training and generating the vote maps. In Random Forests, each training example is labeled into nine classes as described previously. The R1 and R2 used here for Random Forests is 2.8 and 5.6 respectively. This relation of values is set in order to distribute the training data equally in all labels. The Gini criterion is used as the metric to split training data.

In test, for each training neighbour example retrieved for each test contour descriptor, via KNN or Random Forests, two votes are casted. This is done because it is not possible to determine whether the center of the object is in the concave or convex part of the curve. This is done by drawing a tangent to the contour at the salient point of the test descriptor and the both perpendicular directions of the tangent are used to compute the orientation angles. The angles are computed as the angles formed by the perpendicular vectors to the tangent and the x-axis of the image. The x and y displacements from the salient point to the center of the object have to be normalized in orientation with the angles calculated and the rotation matrix. Then each pair of displacements have to be normalized in size too, this is done by dividing the number of pixels of the contour.

In the original approach, the entire boundary of an object is used to construct the descriptor of each salient point. With this approach, we can detect easily isolated moths, but this method can not be able to detect moths touching each other, because the boundary shape of this kind of objects differs in a great extent to the shapes of the training objects that contain a single moth, figure (4.11) is an example of touched moths.

Using the whole contour rises the problem of detecting touching moths. Here we also tested this descriptors using only fragments of contours. In this way, we try to solve the multiple connected moths problem, but the results are poor too. We believe this is because of the distortions created in background removing which makes the detection with boundaries difficult and pieces of contours augment the noise impacts, see figure (4.12).

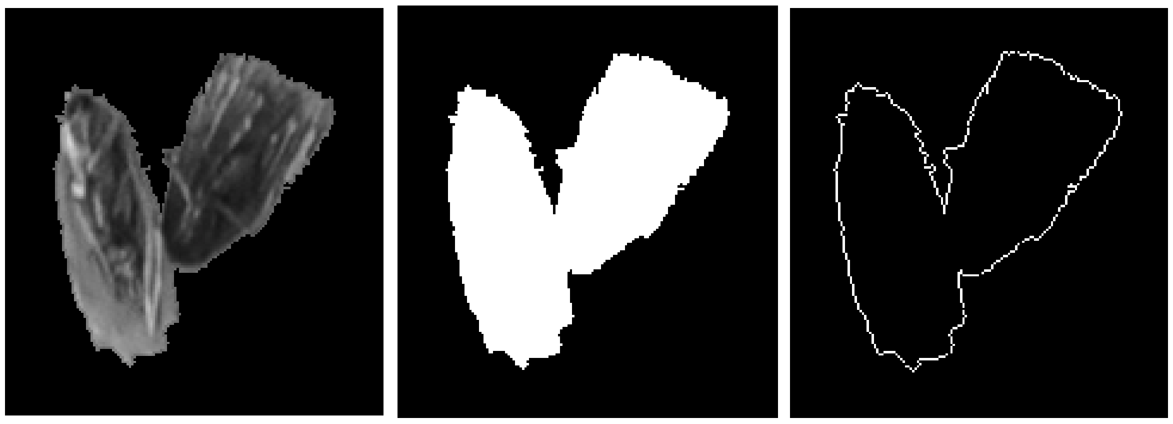


Figure 4.11: Contour extraction of two attached moths, the contour differs to contours of training examples which are single moth in each sample

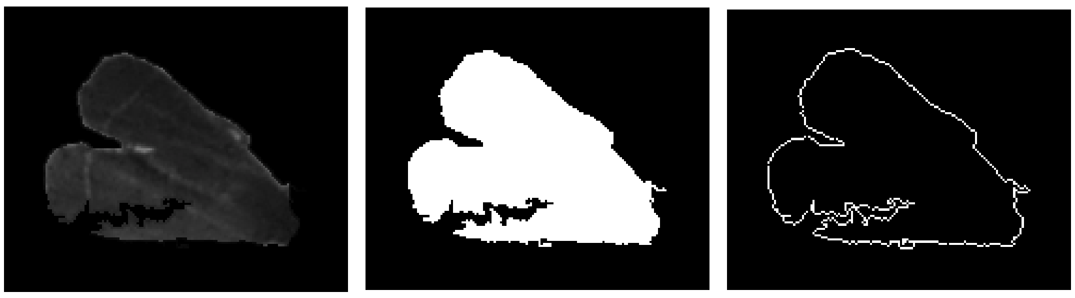


Figure 4.12: Contour extraction of an not complete moth caused by a background removing process

4.2.2 Mesh approach

The mesh detection approach was also tested for this problem. In our experiment, the size of each SIFT descriptor and the distance of its neighbours at horizontal and vertical directions was set to 10 pixels. This gives approximately 100 detections per moth. The orientation of each SIFT descriptor is also treated in this case: The distance of a keypoint to the center of a moth is transformed in orientation and size as the way explained in previous sections.

Random Forests and KNN were tested in conjunction with mesh detection. Both algorithms have similar performances although Random Forests works slightly better. In Random Forests, 9 relative positions are used as the class of each example. For this problem, the distance variable used to classify relative position R1 and R2 are: 2 and 4 for ”near” and ”far” respectively. The same approach as the one used in the UIUC Car experiment is used here: 25 trees are built and trees are grown until the number of examples is less than 10 or all examples are of the same class. We have tested using 50 trees instead 25 obtaining a similar performance. In KNN, we select 30 nearest neighbours for each test example. Similar results are obtained when using different number of neighbours.

Additionally, we have used the approach used in UIUC Car problem: Extract both training and test mesh SIFT points in different sizes. In this way, we try to solve the problem of detecting the centers of bigger moths that are under represented in the training data. With this multiple scale SIFT points approach, we extract SIFT points in mesh pattern with both the size of the SIFT points and the distance between horizontal and vertical neighbours of keypoints varied from 6 to 24. Then 19 voting maps are generated for each image and maxima locations are selected using back-projection and local maxima. After that, a ranking is made, points with higher scores will be included as predictions. Each selected point occupies a rectangle region according to its size in the voting map proportionally to its scale, lower ranked points localized in this region will not be selected.

4.2.3 SIFT approach

We also tried to use SIFT detector for this problem with the same configuration and maxima selections methods described in the mesh approach. But the results shown in Table (4.3) are poor compared to the mesh detection approach.

4.2.4 Class Specific Hough Forests

Class Specific Hough Forests is tested in this problem, but the performance is very poor. The reason is that this algorithm can not handle orientation changes. In the moth detection

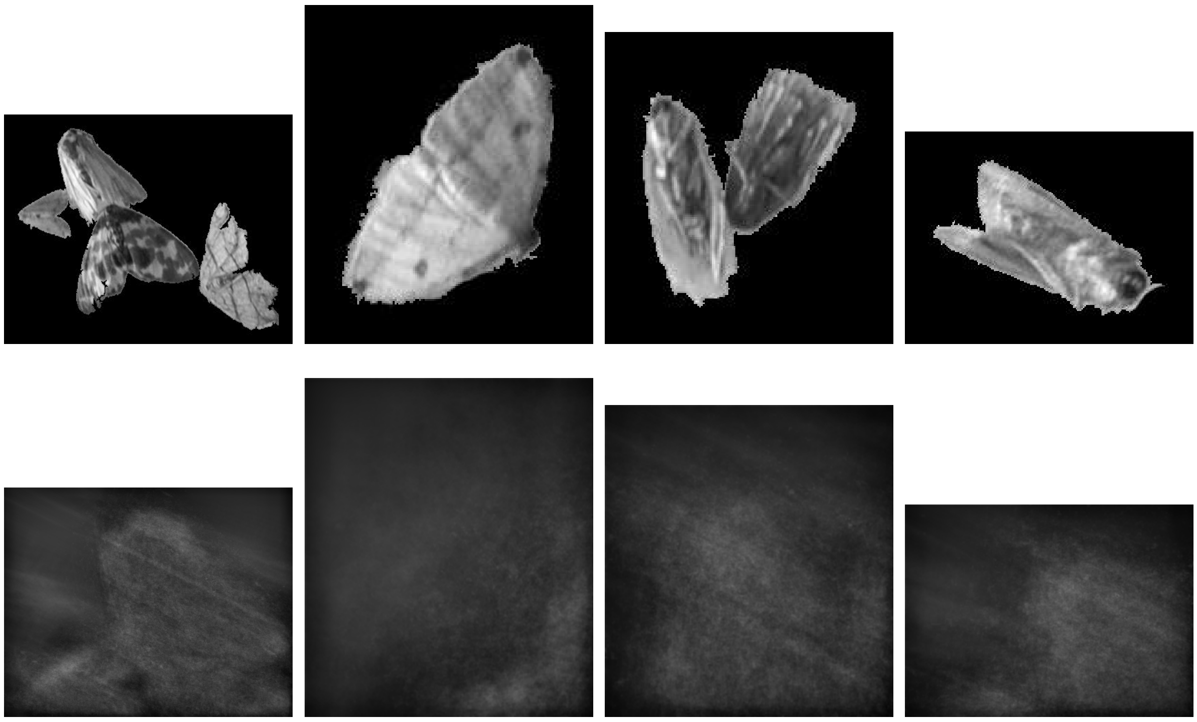


Figure 4.13: The smoothed vote maps (second row) along with their respective original images (first row)

problem (see Figure (4.9)), we can see that each moth has a different orientation in addition to a different scale. In Class Specific Hough Forests, image appearance patch are considered as the descriptor, each of them has 32 rectangles of size 16x16 extracted from different feature channel [Gall and Lempitsky (2009)]. During the training, a binary test is applied to divide different patches in different leaves of a randomized Forests. In test, same binary tests used in training are applied to each patch in order to find its nearest neighbours. Note that the binary test implies selecting randomly 2 locations (P_1 and P_2), and a change of patch orientation makes the positions P_1 and P_2 move to other locations. For this reason, a same patch used in training and detected in test with some orientation variation would output a near random center localization. In figure (**??**), some detection results are shown. The first row shows the original test images and the second row shows the generated smoothed vote maps using Hough forests. It can be observed that there are no clear peaks in the vote maps and the prediction would be almost random.In conclusion, this algorithm is not applicable for this moth center detection task unless all moths have been previously orientated.

4.2.5 Results

Various object detection algorithms have been tested. Table (4.3) shows the result obtained in our experiments. The first four rows indicate the results obtained by mesh based detection approaches for both single and multi-scale problems. We can see that the back-projection based algorithms obtain better performances compared to local maxima based methods. The four following rows present the results obtained using SIFT detection approach. As in the case of UIUC Cars, these methods achieve poor performances comparing to mesh based detection approaches. In the last rows, the results for methods using scaled mesh detections in training are presented. This method does not improve the results obtained using single scale detection in training.

Figure (4.3) shows the Precision-Recall curves of these algorithms for both single and multi-scale problems. The curves of algorithms with good performances are on the top of the figures. The best F-measure peak achieved is near 87%. This is shown in the top right position of the figure.

The results show that the performance of the proposed mesh detection based algorithm is fairly good for this problem and outperforms other tested algorithms. The result detection image with this approach is shown in figure (4.15). The results also confirms that unlike Hough Forests, the proposed algorithm manages to handle the orientation changes of the objects of interest. The algorithm with best performance is the method using the combination of mesh detection, Random Forests and back-projection.

Table 4.3: Performance at F-measure peak for moth center detection problem

Methods	F-measure
Mesh Random Forests + Local maxima	83.89%
Mesh KNN + Local maxima	83.38%
Mesh Random Forests + Back-projection	87.70%
Mesh KNN + Back-projection	85.64%
SIFT Random Forests + Local maxima	52.97%
SIFT KNN + Local maxima	61.28%
SIFT Random Forests + Back-projection	48.63%
SIFT KNN + Back-projection	43.96%
Scaled Mesh Random Forests + Local maxima	78.95%
Scaled Mesh KNN + Local maxima	78.66%
Scaled Mesh Random Forests + Back-projection	74.16%
Scaled Mesh KNN + Back-projection	70.15%

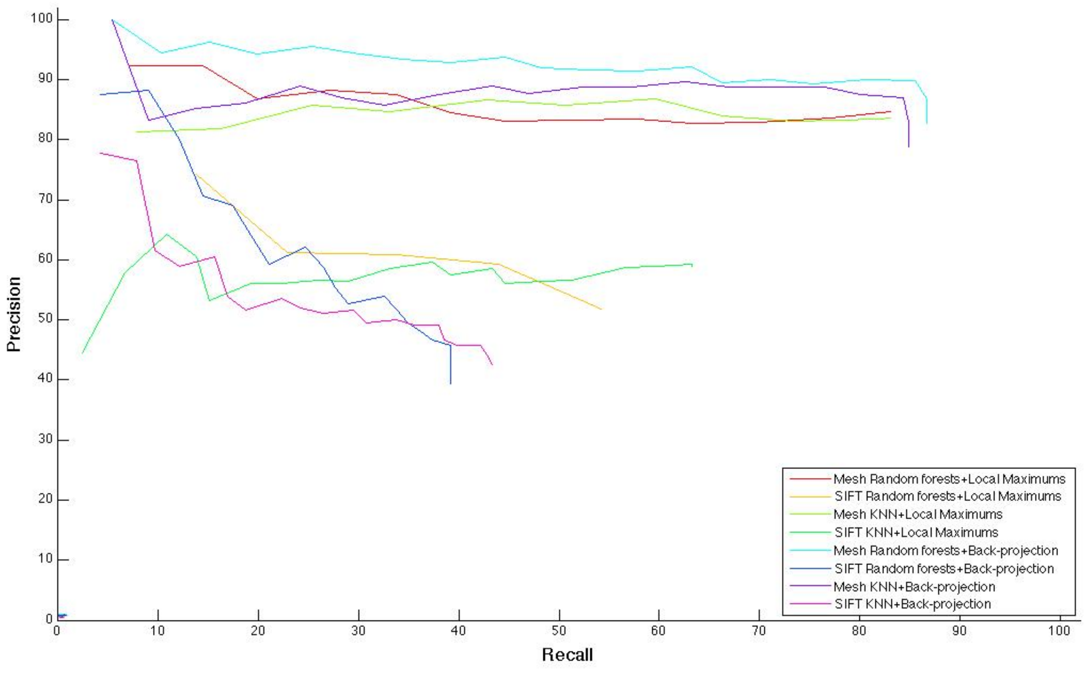


Figure 4.14: Precision-Recall curves for all moth experiments

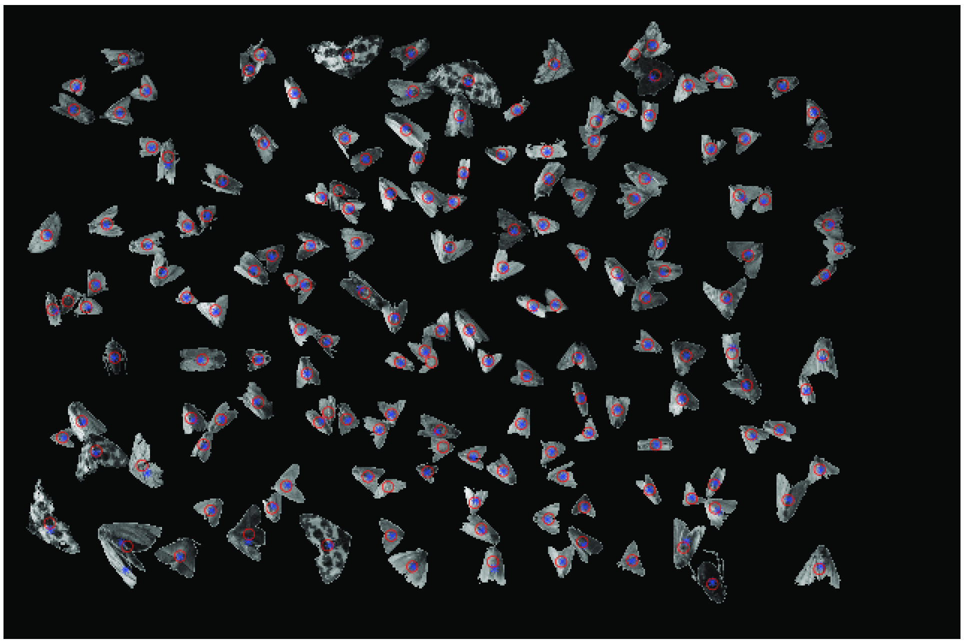


Figure 4.15: A test image with detections indicated in blue stars and expected center regions in a red circle.

The results of mesh based detection approaches show that they are capable to detect the centers of touched moths in an image and are able posteriorly to eliminate the region where the object is, this property can also be used in object segmentation problems. One application of these approaches is to separate touched insects in a biomonitoring system or any kind of touched objects.

SIFT detection based algorithms obtain very poor results for this problem, we believe this is due to the fact that the number of detections obtained using SIFT detector is very low when compared with the mesh approach. The scaled mesh based detection does not approaches obtain good results too.

Some detection results are shown in figure (4.16), In the first row, the correct detections are shown, they are some of the most difficult cases in the problem due to the proximity between moths. In last two rows, some error examples are shown. Most of them are moths of a minority classes which have their wings spread and are bigger than others. If we increase the selection threshold, those false positives will be removed from the final rank, but new false negatives well be created too. We believe that with more training examples of this type of moths the error rating may decrease.

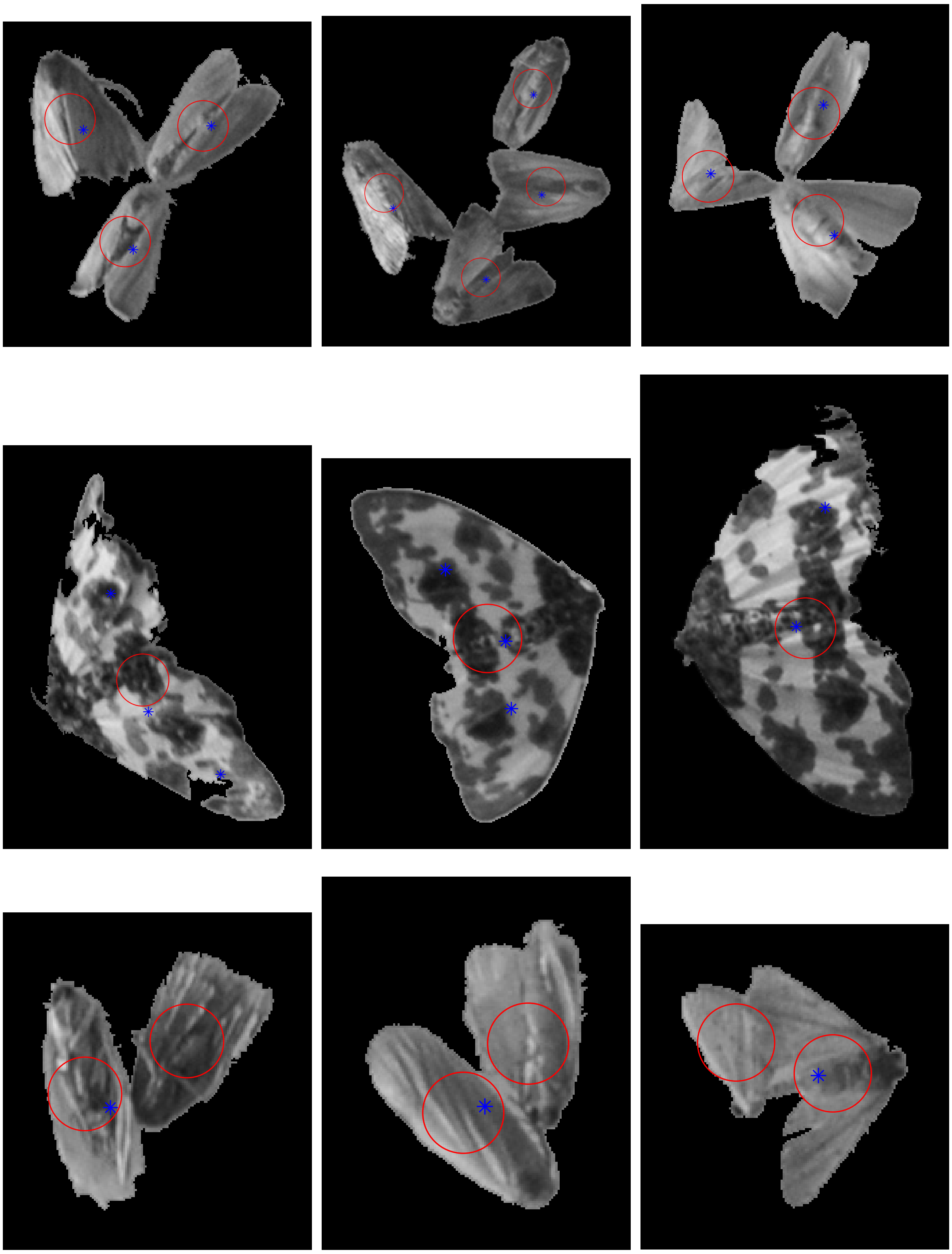


Figure 4.16: Moth detection examples. First row shows the the correct detections of some of the most difficult cases. Last two rows show the effect of a prediction selection threshold, a higher threshold will cause false negatives (second row)and a lower threshold will includes mores false positives in the final ranking (third row).

Chapter 5

Conclusions and future work

A good object detection algorithm has to be effective in different image conditions such as: partial occlusion, scale, illumination and orientation changes. The main contribution of this work is the proposal of a new generic object detection algorithm which obtains good performance for detecting any kind of objects under such variations. The main idea of the proposed method is to learn the relative displacements of each object feature descriptor to the center of the object. In test, for each test descriptor its most similar training descriptors and their displacements are recovered, these displacements are used to vote the center of the object. In order to handle object orientation and scale variations, the training displacements are not stored in image coordinates but in relation to the patch coordinates. The stored coordinates are rotated and scaled according to the patch orientation and scale. These transformed coordinates are learned using a Random Forests or KNN model and for the Random Forests are stored in the leaves of the trees. In test, for each new descriptor the Random Forests or KNN model returns the most similar training displacements. Those displacements are transformed back into the test patch coordinate system and a vote is casted. The votes are accumulated into a voting map. And the maxima in this map indicate the locations of the objects of interest.

The proposed algorithm is a generic object detection method that can in principle be applied to the detection of any type of object. The proposed algorithm is based on the feature descriptor used. It has to be orientation and scale invariant. The performance depends on how discriminative is the feature descriptor. This confers the algorithm with capacity to handle changes in illumination, object rotation, scale, etc. It has to be noted that the proposed algorithm is not invariant to objects that present view point changes.

In order to test the performance of the proposed algorithm several experiments have been carried out. Several combinations of methods are tested in UIUC-Car [Agarwal *et al.* (2004); Agarwal and Roth (2002)] and Moth [Yao *et al.* (2013)] problems. The proposed approach works well in object detection when combined with mesh detection. Mesh detection computes a grid of detections uniformly for the whole image. This approach performs better than the state-of-the-art algorithm Hough Forests in both problems. In moth problem, the proposed algorithm obtains rather good results. On the other hand, Hough Forests is not capable to detect moths mainly because the moths are displayed in different orientations. This proves the effectiveness of the proposed algorithm. Additionally, the proposed algorithm was computationally faster than Hough Forests in our experiments. Although, we did not do specific experiments to evaluate the computational performance of the different algorithms. We also tested other variations of the proposed approach. We tested using SIFT detector instead of mesh detection, there is a clear deterioration of performance. We believe that this is due to the fact that a much smaller number of detections are obtained with SIFT with respect to mesh detection. We also tested Beam Angle descriptor in the moths problem, but the results are also poor, the reason is that this dataset contains touching moths and incomplete moths which makes the extracted contour descriptors unstable.

Concerning the future works, some variant algorithms could be experimented to improve the performance of the system. As we mentioned before, the proposed system is divided into different parts which confers the system with a modular structure. Each one of these modules can be replaced by other methods for different problems or situation. For example, in our experiments, we tested using Beam Angle descriptor instead of SIFT descriptor to capture contour feature in the moth problem. Some other possible modifications could be worth trying:

- The use of Random Forests without discretizetion or label annotation, this is, training Random Forests for regression using the displacements directly to split samples.
- To combine both contour based descriptors and patch based descriptors to improve the system performance.
- To include a descriptor that could handle view point variation such as Hessian or Harris affine descriptor.
- To combine SIFT descriptors of different sizes both in training and test, this is to use SIFT descriptors of different size levels of an image and include all of them in both training and test data. In this way, different aspects of the object of interest could be handled.

Other future lines of research related to this work that we are studying are: to modify Hough Forests to get a orientation invariant version, this can be done by adapting an orientation invariant descriptor in the algorithm; to adapt this detection method in a classification algorithm, this is, creating several class specific detection models using this approach to classify images according to the containing objects.

References

Shivani Agarwal & Dan Roth. Learning a sparse representation for object detection. In *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ECCV '02, pages 113–130, London, UK, UK, 2002. Springer-Verlag.

Shivani Agarwal, Aatif Awan, & Dan Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(11):1475–1490, November 2004.

K. Anderson & P. W. McOwan. A real-time automated system for the recognition of human facial expressions. *Trans. Sys. Man Cyber. Part B*, 36(1):96–105, February 2006.

Nafiz Arica & Fatos T. Yarman Vural. Bas: a perceptual shape descriptor based on the beam angle statistics. *Pattern Recogn. Lett.*, 24(9-10):1627–1639, June 2003.

Nafiz Arica & FatoŞ T. Yarman-Vural. A compact shape descriptor based on the beam angle statistics. In *Proceedings of the 2nd international conference on Image and video retrieval*, CIVR'03, pages 152–162, Berlin, Heidelberg, 2003. Springer-Verlag.

Jeffrey S. Beis & David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1000–, Washington, DC, USA, 1997. IEEE Computer Society.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

Oren Boiman, Eli Shechtman, & Michal Irani. In defense of nearest-neighbor based image classification. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society, 2008.

Gunilla Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(6):849–865, November 1988.

A. Bosch, A. Zisserman, & X. Munoz. Image classification using random forests and ferns. In *Proceedings of the 11th International Conference on Computer Vision, Rio de Janeiro, Brazil*, 2007.

Cynthia Breazeal & Brian Scassellati. A context-dependent attention system for a social robot. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 1146–1151, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

Rich Caruana & Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. of the 23rd International Conference on Machine Learning*, pages 161–168, New York, NY, USA, 2006. ACM Press.

Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, & Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

Navneet Dalal & Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.

Yigithan Dedeoglu. Moving object detection, tracking and classification for smart video surveillance, 2004.

R. Fergus, P. Perona, & A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Madison, Wisconsin, June 2003.

Vittorio Ferrari, Frédéric Jurie, & Cordelia Schmid. Accurate object detection with deformable shape models learnt from images. In *Conference on Computer Vision & Pattern Recognition*, IEEE, jun 2007.

Yoav Freund & Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. of the 13th International Conf. on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

Yoav Freund & Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

Juergen Gall & Victor Lempitsky. Class-specific hough forests for object detection. In *In Proceedings IEEE Conference Computer Vision and Pattern Recognition*, 2009.

Juergen Gall, Angela Yao, Nima Razavi, Luc Van Gool, & Victor Lempitsky. Hough forests for object detection, tracking, and action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(11):2188–2202, November 2011.

Alvaro Garcia-Martin & Jose M. Martinez. Robust real time moving people detection in surveillance scenarios. In *Proceedings of the 2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, AVSS '10, pages 241–247, Washington, DC, USA, 2010. IEEE Computer Society.

L. K. Hansen & P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, October 1990.

Chris Harris & Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

Christoph H. Lampert, Matthew B. Blaschko, & Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *In Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR*, pages 1–8, 2008.

Svetlana Lazebnik & Cordelia Schmid. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *In CVPR*, pages 2169–2178, 2006.

Bastian Leibe & Bernt Schiele. Interleaved object categorization and segmentation. In *In BMVC*, pages 759–768, 2003.

Bastian Leibe, Ales Leonardis, & Bernt Schiele. Combined object categorization and segmentation with an implicit shape model. In *In ECCV workshop on statistical learning in computer vision*, pages 17–32, 2004.

Bastian Leibe, Aleš Leonardis, & Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vision*, 77(1-3):259–289, May 2008.

Vincent Lepetit, Pascal Lagger, & Pascal Fua. Randomized trees for real-time keypoint recognition. In *In CVPR*, pages 775–781, 2005.

Kobi Levi & Yair Weiss. Learning object detection from a small number of examples: the importance of good features. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'04, pages 53–60, Washington, DC, USA, 2004. IEEE Computer Society.

Fengjie Li, Wei Tong, Rong Jin, Anil K. Jain, & Jung-Eun Lee. An efficient key point quantization algorithm for large scale image retrieval. In *Proceedings of the First ACM workshop on Large-scale multimedia retrieval and mining*, LS-MMRM '09, pages 89–96, New York, NY, USA, 2009. ACM.

Jefrey Lijffijt, Panagiotis Papapetrou, Kai Puolamäki, & Heikki Mannila. Analyzing word frequencies in large text corpora using inter-arrival times and bootstrapping. In *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part II*, ECML PKDD'11, pages 341–357, Berlin, Heidelberg, 2011. Springer-Verlag.

Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30:79–116, 1998.

David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

David G. Lowe. Local feature view clustering for 3d object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 682–688. Springer, 2001.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

Subhransu Maji & Jitendra Malik. Object detection using a max-margin hough transform. In *CVPR*, pages 1038–1045. IEEE, 2009.

G. Martínez-Muñoz & A. Suárez. Switching class labels to generate classification ensembles. *PATTERN RECOGNITION*, 38(10):1483–1494, 2005.

Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, & Alberto Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 31(2):1, 2009.

Gonzalo Martínez-Muñoz, Natalia Larios Delgado, Eric N. Mortensen, Wei Zhang 0014, Asako Yamamuro, Robert Paasch, Nadia Payet, David A. Lytle, Linda G. Shapiro, Sinisa Todorovic, Andrew Moldenke, & Thomas G. Dietterich. Dictionary-free categorization of very similar objects via stacked evidence trees. In *CVPR*, pages 549–556. IEEE, 2009.

K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, & L. Van Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, November 2005.

Krystian Mikolajczyk. Multiple object class detection with a generative model. In *In CVPR*, pages 26–36, 2006.

Frank Moosmann, Bill Triggs, & Frederic Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *In NIPS*, 2007.

Frank Moosmann, Eric Nowak, & Frederic Jurie. Randomized clustering forests for image classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1632–1646, September 2008.

Marius Muja & David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

Jim Mutch & David G. Lowe. Multiclass object recognition with sparse, localized features. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 11–18, Washington, DC, USA, 2006. IEEE Computer Society.

Eric Nowak, Frédéric Jurie, & Bill Triggs. Sampling strategies for bag-of-features image classification. In *European Conference on Computer Vision*. Springer, 2006.

Andreas Opelt, Axel Pinz, & Andrew Zisserman. A boundary-fragment-model for object detection. In *Proceedings of the 9th European conference on Computer Vision - Volume Part II*, ECCV'06, pages 575–588, Berlin, Heidelberg, 2006. Springer-Verlag.

Andreas Opelt, Axel Pinz, & Andrew Zisserman. Learning an alphabet of shape and appearance for multi-class object detection. *Int. J. Comput. Vision*, 80(1):16–44, October 2008.

S. Savarese, J. Winn, & A. Criminisi. Discriminative object class models of appearance and shape by correlatons. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2033–2040, Washington, DC, USA, 2006. IEEE Computer Society.

Henry Schneiderman & Takeo Kanade. A statistical model for 3d object detection applied to faces and cars. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2000.

Jamie Shotton, Andrew Blake, & Roberto Cipolla. Contour-based learning for object detection. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01*, ICCV '05, pages 503–510, Washington, DC, USA, 2005. IEEE Computer Society.

Leo Breiman Statistics & Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

3 Frederic Jurie4 Liu Yang1 Rong Jin1 Rahul Sukthankar2. Unifying discriminative visual codebook generation with classifier training for object category recognition. In *CVPR*, 2008.

Antonio Torralba, Kevin P. Murphy, & William T. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(5):854–869, May 2007.

Paul Viola & Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

Wang Wei, Hong Jun, & Tang Yiping. Image matching for geomorphic measurement based on sift and ransac methods. In *International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 2: Software Engineering, December 12-14, 2008, Wuhan, China*, pages 317–320. IEEE Computer Society, 2008.

Wikipedia. Cbir system; http://en.wikipedia.org/wiki/content-based_image_retrieval, 2004.

Wikipedia. Lab color space; http://en.wikipedia.org/wiki/lab_color_space, 2004.

Wikipedia. Laplacian of gaussian; http://en.wikipedia.org/wiki/blob_detection, 2004.

Wikipedia. Mixture model; http://en.wikipedia.org/wiki/mixture_model, 2004.

Jutta Willamowski, Damian Arregui, Gabriella Csurka, Christopher R. Dance, & Lixin Fan. Categorizing nine visual classes using local appearance descriptors. In *In ICPR Workshop on Learning for Adaptable Visual Systems*, 2004.

J. Winn, A. Criminisi, & T. Minka. Object categorization by learned universal visual dictionary. In *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, ICCV '05, pages 1800–1807, Washington, DC, USA, 2005. IEEE Computer Society.

Haim J. Wolfson. Generalizing the generalized hough transform. *Pattern Recogn. Lett.*, 12(9):565–573, September 1991.

Jianchao Yang, Kai Yu, Yihong Gong, & Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *in IEEE Conference on Computer Vision and Pattern Recognition(CVPR*, 2009.

Qing Yao, Qingjie Liu, Thomas G. Dietterich, Sinisa Todorovic, Jeffrey Lin, Guangqiang Diao, Baojun Yang, & Jian Tang. Segmentation of touching insects based on optical flow and {NCuts}. *Biosystems Engineering*, 114(2):67 – 77, 2013.

Wei Zhang & Thomas G. Dietterich. Learning visual dictionaries and decision lists for object recognition, 2008.