

Chapter 2

Marginal Space Learning

2.1 Introduction

Automatic detection of an anatomical structure (object) in medical images is a prerequisite for subsequent tasks such as recognition, segmentation, measurement, or motion tracking, and therefore has numerous applications. The goal of the detection is to estimate the position, orientation, and size of the target anatomical structure. The pose parameters can be represented as an oriented bounding box (to distinguish from an axis-aligned bounding box).

Recently, discriminative learning based approaches have been proved to be efficient and robust to detect 2D objects [14,31]. In these methods, object detection is formulated as a classification problem: whether an image window contains the target object or not [31]. During object detection, the pose parameter space is quantized into a large set of discrete hypotheses and each hypothesis is tested by a trained classifier to get a detection score. The hypotheses with the highest score are taken as the detection results. Exhaustive search for the best hypothesis makes the system robust under local optima. This search strategy is quite different from other parameter estimation approaches, such as deformable models, where an initial estimate is adjusted using the gradient descent techniques to optimize a predefined objective function.

There are two challenges to extend learning based object detection approaches to 3D. First, the number of hypotheses increases exponentially with respect to the dimensionality of the parameter space. Although the dimensionality of the image space only increases by one from 2D to 3D, the dimensionality of the pose parameter space increases from 5 to 9. For 2D object detection, there are five unknown parameters to be estimated or searched for from an input image, namely, translation in two directions, one rotation angle, and two scales. For 3D object detection, there are nine degrees of freedom for the anisotropic similarity transformation, namely three translation parameters, three rotation angles, and three scales. Note that the ordinary similarity transformation defines only isotropic scaling, corresponding to one scale parameter. However, to better cope with nonrigid deformations of the

target object, we use anisotropic scales. The number of hypotheses is an exponential function of the dimensionality of the pose parameter space. If each dimension is quantized to n discrete values, the number of hypotheses is n^9 . For a very coarse estimation with $n = 10$, $n^9 = 1,000,000,000$ for 3D instead of $n^5 = 100,000$ for 2D. As a result, the computational demands for the 3D case present a challenge to the current desktop computers, to provide the testing results in a reasonable time. Even a five dimensional pose parameter space for a 2D object is already too large to achieve real-time performance. Note that for the 2D face detection example in [31], they only searched a three dimensional pose parameter space, two dimensions for position and one dimension for isotropic scaling, by constraining the face in an up-straight front view with a fixed aspect ratio.

The second challenge of extending the learning based object detection to 3D is that we need efficient features to search the orientation space. To perform a classification to an orientation hypothesis, the image features should be a function of the orientation hypothesis. Since we want to explicitly estimate the orientation of an object, rotation invariant features cannot be applied. There are two ways to embed the orientation information into image features, rotating either the feature template or the volume. Haar wavelet features can be efficiently computed under translation and scaling using the integral images [24, 31], but no efficient ways are available to rotate the Haar wavelet features. For 2D object detection, there is only one degree of freedom in rotation. It is possible to discretize orientation into a small number of categories, e.g., 10° of incremental rotation in $[0, 360^\circ)$, resulting in 36 orientations. The input image is rotated accordingly for each orientation category to generate a set of rotated images. A detector is trained under one fixed orientation and is applied to all the rotated images to detect an object with different orientations [8]. However, a 3D volume contains much more data; therefore, it is very time consuming to rotate the volume. The computation time to rotate a volume with $512 \times 512 \times 512$ voxels is equivalent to rotate 512 images each with 512×512 pixels. Furthermore, there are three degrees of freedom in 3D rotations. To cover the full orientation space with a sampling resolution of 9.72° , we need 7416 discrete orientation hypotheses [18]. Since volume rotation is time consuming, it is impractical to perform thousands of volume rotations for orientation estimation.

This chapter presents solutions to the two challenges discussed above. First, it introduces an efficient 3D learning-based object detection method, called Marginal Space Learning (MSL). The idea of MSL is to avoid learning in the full similarity transformation space by incrementally learning classifiers in marginal spaces of lower dimensions. The estimation of an object's pose is split into three problems: position estimation, position-orientation estimation, and position-orientation-scale estimation. This incremental learning approach contributes to a highly efficient object detection paradigm. Second, we introduce the steerable features, as a mechanism to search the orientation space, thus avoiding expensive volume/image rotations. The idea is to sample points (voxels) from a given volume under a sampling pattern that embeds the position, orientation, and scale information of a pose hypothesis. Each sample point is associated with a set of local features such

as local intensity and gradient. The efficiency of steerable features comes from the fact that much fewer points (defined by the sampling pattern) are needed for manipulation, in comparison to the whole volume.

The remainder of this chapter is organized as follows. In Sect. 2.2, we present the whole MSL workflow, including the derivation of the pose parameter ground truth from a training set with annotated meshes, training of each individual pose parameter estimator, and aggregation of multiple pose candidates to achieve a consolidated estimate of the object. We then discuss implementation details, introducing 3D image features in Sect. 2.3 and the Probabilistic Boosting-Tree (PBT) classifier in Sect. 2.4. In Sect. 2.5, we use automatic 3D heart chamber detection in CT volumes as an example to demonstrate the efficiency and robustness of MSL. In Sect. 2.6, we extend the MSL principle to directly estimate nonrigid deformation parameters to further improve the shape initialization accuracy for nonrigid object segmentation. In Sect. 2.7, we provide theoretical justifications of MSL and link it to the shortest path computation in graph search. The MSL is shown to be an efficient breadth-first beam search in the posterior probability of the pose parameter space. This chapter concludes with Sect. 2.8.

2.2 3D Object Detection Using Marginal Space Learning

For an in-depth understanding of this section, we refer the reader to earlier learning based object detection publications [14, 29, 31].

2.2.1 *Derivation of Object Pose Ground Truth From Mesh*

To train object detection classifiers, we need a set of 3D volumes, called the training set. The volumes in the training set are typically converted to a low isotropic resolution (e.g., 3 mm). For each volume, we need a nine dimensional vector of the ground truth about the position, orientation, and size of the target object in the volume. These nine pose parameters can be visually represented as a bounding box of the object. In some applications, the pose parameters are readily available from the annotation of the image data. This is especially common for 2D object detection since it is easy to draw a bounding box of the target object. However, drawing a 3D bounding box aligned with the orientation of the 3D object is not trivial. It is more convenient to annotate a few landmarks of the object and derive a box from the landmarks. Alternatively, since in many applications, the accurate object segmentation is the ultimate goal, a 3D surface mesh is annotated by experts for each volume in the training set, either manually or semi-automatically. In the following, we present an ad-hoc method to derive the 3D pose parameters/bounding box of a surface mesh. This is an intuitive solution, but by no means optimal.

In Chap. 6, we present a more theoretically founded solution to derive optimal nine pose parameters from a set of 3D meshes to minimize the mesh initialization error after pose estimation.

The object orientation cannot be easily derived from a 3D mesh. Different methods are often demanded to define the orientation of different target objects. Many anatomies have an intrinsic, well-accepted orientation definition. For example, the orientation of the heart chambers, defined by the long axis and short axis, is documented in detail by the echocardiography community [17]. In the heart chamber segmentation application [33, 34], we use the long axis of a chamber as the z axis. The perpendicular direction from a predefined anchor point to the long axis defines axis x . For different chambers, we have freedom to select the anchor point, as long as it is consistent. For example, for the left ventricle, we can use the aortic valve center as the anchor point. The third axis y is the cross-product of axes z and x . A 3×3 rotation matrix \mathbf{R} is determined using the x , y , and z axis as the first, second, and last column of \mathbf{R} , respectively. An orientation hypothesis is represented as three Euler angles, ψ^t , ϕ^t , and θ^t , which can be derived from the rotation matrix \mathbf{R} using the following relationship

$$\mathbf{R} = \begin{bmatrix} \cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & \cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & \sin \psi \sin \theta \\ -\sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & -\sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & \cos \psi \sin \theta \\ \sin \theta \sin \phi & -\sin \theta \cos \phi & \cos \theta \end{bmatrix}. \quad (2.1)$$

We then calculate a bounding box aligned with the object-oriented local coordinate system for the mesh points. The bounding box center gives us the position ground truth X^t , Y^t , and Z^t , and the box size along each side defines the ground truth of scaling S_x^t , S_y^t , and S_z^t , respectively.

For object segmentation, we also need to calculate the mean shape from the training set so that after object pose estimation we can align the mean shape to get an initial estimate of the true shape. For a target object, using the above bounding box based method, we calculate its position ($\mathbf{T} = [X, Y, Z]^t$), orientation (represented as a rotation matrix \mathbf{R}), and anisotropic scaling ($\mathbf{S} = [S_x, S_y, S_z]^t$). We then transform each point from the world coordinate system, \mathbf{M}_{world} , to the object-oriented coordinate system, \mathbf{m}_{object} , and calculate the average over the whole training set to get a mean shape. Here, we assume that the training shapes have intrinsic mesh point correspondence, namely, each mesh has the same number of points and the same point index in different meshes corresponds to the same anatomy. The transformation between \mathbf{M}_{world} and \mathbf{m}_{object} is

$$\mathbf{M}_{world} = \mathbf{R} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \mathbf{m}_{object} + \mathbf{T}. \quad (2.2)$$

Reversing the transformation, we can calculate the position in the object-oriented coordinate system as

$$\mathbf{m}_{object} = \begin{bmatrix} \frac{1}{\bar{s}_x} & 0 & 0 \\ 0 & \frac{1}{\bar{s}_y} & 0 \\ 0 & 0 & \frac{1}{\bar{s}_z} \end{bmatrix} \mathbf{R}^{-1} (\mathbf{M}_{world} - \mathbf{T}). \quad (2.3)$$

The mean shape is the average over the whole training set

$$\bar{\mathbf{m}} = \frac{1}{N} \sum_{i=1}^N \mathbf{m}_{object}^i, \quad (2.4)$$

where N is the number of training samples.

2.2.2 Principle of Marginal Space Learning

Figure 2.1 shows the basic idea of machine learning based 3D object detection. First, we train a classifier, which assigns a score in the range $[0, 1]$ to each input hypothesis about the object pose. We then quantize the full pose parameter space into a large number of hypotheses. Depending on the quantization resolution, the number of hypotheses can easily reach an order over 1 billion. Each hypothesis is tested with the classifier to get a score. Based on the classification scores, we select the best one or several hypotheses. We may need to aggregate multiple best hypotheses into a final single detection result. Unlike the gradient based search in deformable models or Active Appearance Models (AAM) [4], the classifier in this framework acts as a black box without an explicit closed-form objective function.

As we discussed, one drawback of the learning based approach is that the number of hypotheses increases exponentially with respect to the dimensionality of the parameter space. Nevertheless, in many applications, the posterior distribution is clustered in a small region in the high dimensional parameter space. Therefore, the uniform and exhaustive search is not necessary and wastes the computational power.

The MSL is an efficient method to partition such parameter space, by gradually increasing the dimensionality of the search space. Let Ω be the space where the solution to the given problem exists and let P_Ω be the true probability distribution that needs to be learned. The learning and computation are performed in a sequence of marginal spaces

$$\Omega_1 \subset \Omega_2 \subset \dots \subset \Omega_n = \Omega \quad (2.5)$$

such that Ω_1 is a low dimensional space (e.g., three-dimensional translation instead of nine-dimensional similarity transformation), and for each k , $\dim(\Omega_k) - \dim(\Omega_{k-1})$ is small. A search in the marginal space Ω_1 using the learned probability model finds a subspace $\Pi_1 \subset \Omega_1$ containing the most probable values and discards the rest of the space. The restricted marginal space Π_1 is then extended to $\Pi_1^e = \Pi_1 \times X_1 \subset \Omega_2$. Another stage of learning and testing is performed on Π_1^e obtaining

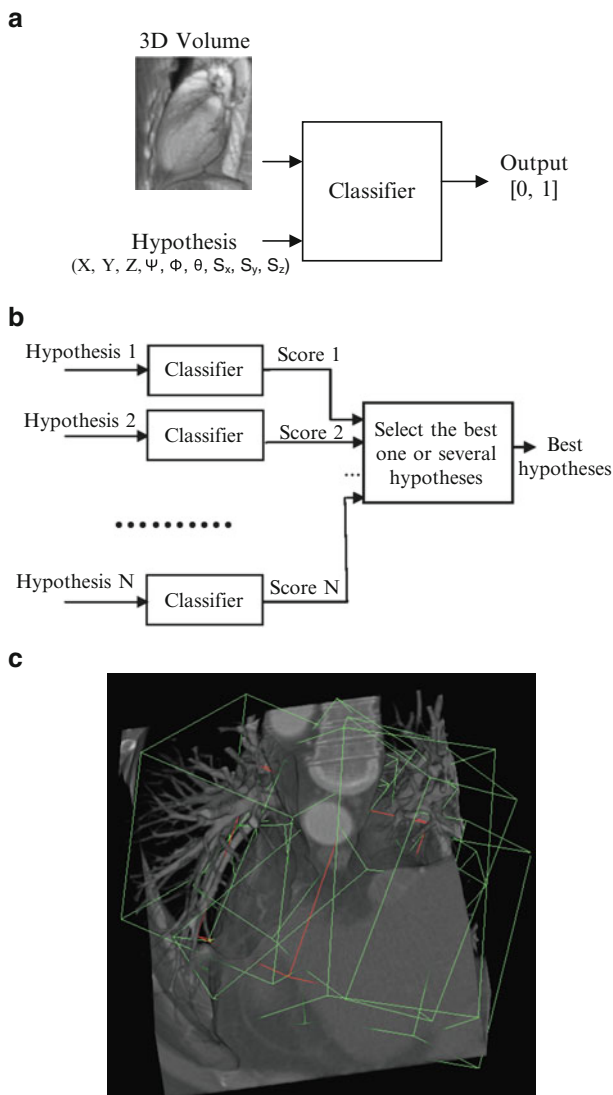


Fig. 2.1 The basic idea of a machine learning based 3D object detection. **(a)** A trained classifier assigns a score to a pose hypothesis. **(b)** The pose parameter space is quantized into a large number of discrete hypotheses and the classifier is used to select the best hypotheses through exhaustive search. **(c)** A few pose hypotheses of the left ventricle (represented as boxes) embedded in a CT volume. ©2008 IEEE. Reprinted, with permission, from Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Four-chamber heart modeling and automatic segmentation for 3D cardiac CT volumes using marginal space learning and steerable features. *IEEE Trans. Medical Imaging* 27(11), 1668–1681 (2008)

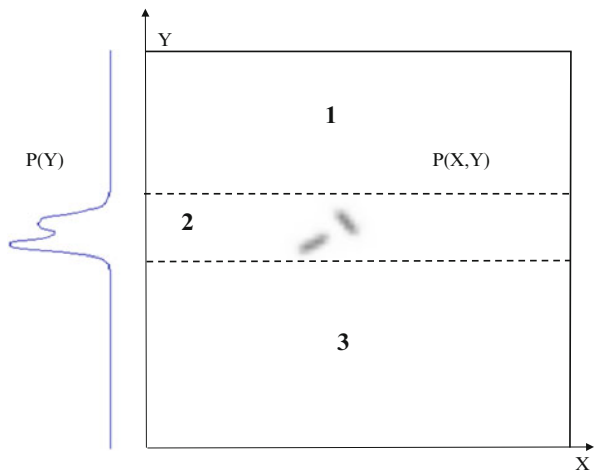


Fig. 2.2 Marginal space learning to search the peaks of the joint distribution $p(x,y)$. A classifier trained on a marginal distribution $p(y)$ can quickly eliminate a large portion (regions 1 and 3) of the search space. Another classifier is then trained on a restricted space (region 2) for the joint distribution $p(x,y)$. ©2008 IEEE. Reprinted, with permission, from Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Four-chamber heart modeling and automatic segmentation for 3D cardiac CT volumes using marginal space learning and steerable features. *IEEE Trans. Medical Imaging* **27**(11), 1668–1681 (2008)

a restricted marginal space $\Pi_2 \subset \Omega_2$ and the procedure is repeated until the full space Ω is reached. At each step, the restricted space Π_k is one or two orders of magnitude smaller than $\Pi_{k-1} \times X_k$. This results in a very efficient algorithm with minimal loss in performance.

Figure 2.2 illustrates a simple example for 2D space search. A classifier trained on $p(y)$ can quickly eliminate a large portion of the search space. We can then train a classifier in a much smaller region (region 2 in Fig. 2.2) for the joint distribution $p(x,y)$. Note that MSL is significantly different from a classifier cascade [31]. In a cascade the learning and search are performed in the same space while for MSL the learning and search space is gradually increased.

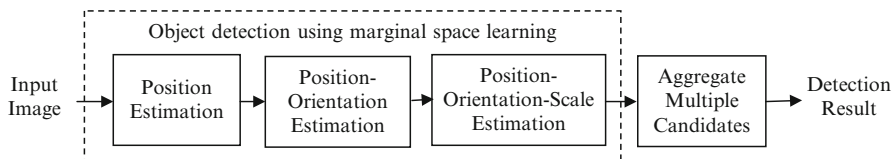


Fig. 2.3 Diagram for 3D object detection using marginal space learning

Let us describe now the general idea of the MSL for 3D object detection. As shown in Fig. 2.3, we split 3D object detection into three steps: position estimation, position-orientation estimation, and position-orientation-scale estimation. The

searching order of the subspaces is based on the following considerations. The position of the object in a volume is the most important information, so it should be determined first. The orientation specifies a local object-oriented coordinate system and the scale of the object is defined along the axes of the coordinate system. Since the scale is defined based on a specified orientation, it is estimated after orientation. After each step we keep a limited number of candidates to reduce the search space.

Besides significantly reducing the search space, another advantage of the MSL is that we can use different features or learning methods to estimate the pose parameters in each step. For example, in position estimation, since we treat rotation as an intra-class variation, we can use the efficient 3D Haar wavelet features. In the following steps of position-orientation estimation and position-orientation-scale estimation, we use steerable features, which are efficient to calculate under rotations. Although in our experiments, the same classifier, i.e., the Probabilistic Boosting-Tree (PBT) classifier [28], is used for all estimation steps, it is possible to use different classifiers for different steps.

2.2.3 Training of Position Estimator

To train a classifier, we need to split hypotheses into two groups, positive and negative, based on their distance to the ground truth. The error in object position and scale estimation is not comparable with that of orientation estimation. Therefore, a search-step-normalized distance measure is defined by normalizing the error in each dimension to the corresponding search step size,

$$E = \max_{i=1,\dots,D} |P_i^e - P_i^t| / \text{SearchStep}_i, \quad (2.6)$$

where P_i^e is the estimated value for parameter i ; P_i^t is the corresponding ground truth; and D is the dimension of the parameter space. For 3D similarity transformation estimation, the pose parameter space is nine dimensional, $D = 9$. A sample is regarded as a positive one if $E \leq 1.0$ or a negative one if $E > 2.0$. The distance from a hypothesis to the ground truth takes a continuous value. It is difficult to draw a clear line to separate positive and negative samples. We intentionally discard samples with a normalized distance between 1.0 and 2.0 to avoid confusing the classifiers.

During the position estimation step, learning is constrained in a marginal space with three dimensions. Given a hypothesis (X, Y, Z) , the classification problem is formulated as whether there is an object centered at (X, Y, Z) . Haar wavelet features are fast to compute and have been shown to be effective for many applications [24, 31]. Therefore, we extended the Haar wavelet features to 3D to be used for learning in this step. Please refer to Sect. 2.3.1 for more information about Haar wavelet features. Note that we tried a position estimator using steerable

features and its performance was slightly worse than the Haar wavelet features on some applications, an expected result.

The search step for position estimation is one voxel. According to Eq. (2.6), a positive sample (X, Y, Z) should satisfy

$$\max\{|X - X^t|, |Y - Y^t|, |Z - Z^t|\} \leq 1 \text{ voxel}, \quad (2.7)$$

where (X^t, Y^t, Z^t) is the ground truth of the object center. Normally, there are $2^3 = 8$ position candidates for each training volume satisfying Eq. (2.7) if the position ground truth X^t , Y^t , and Z^t are not on the imaging grid (e.g., $X^t = 45.2$ voxels). If the ground truth is lying on the imaging grid (e.g., $X^t = 45$ voxels), we may have up to $3^3 = 27$ positive position hypotheses for each training volume.

A negative sample is one with

$$\max\{|X - X^t|, |Y - Y^t|, |Z - Z^t|\} > 2 \text{ voxels}. \quad (2.8)$$

Normally, we have far more negative samples than positive ones. The classifier should have the capability to handle such a significantly skewed distribution of positive and negative samples; otherwise, we have to randomly sample roughly the same number of negative samples to match the positive samples.

The Probabilistic Boosting-Tree (PBT) [28] classifier used in most of our experiments has no difficulty to handle uneven distributions of positive and negative samples; therefore, it does not impose constraints on the cardinality of the sample sets.

However, in some applications we still have to subsample the negative samples because of the memory constraints of a desktop computer. On a 32-bit Windows operating system, the maximum amount of memory a program can use is bounded to 2 GB and can be increased to 3 GB with a special setting of Windows, which still limits the number of negative samples that can be loaded into memory. Normally, we train the classifiers on a 64-bit Windows systems with sufficient amount of memory (we tried up to 72 GB of memory). In addition, the memory constraint can be further alleviated by using a computer cluster or a connection to a cloud system.

We often set an upper limit for negative samples (e.g., 10 million) since we need a reasonable speed for the training process. Our training software first gets a rough estimate of the total number of negative samples. If the result is smaller than the upper limit, no sub-sampling is applied; otherwise, we randomly subsample the negative samples to match that limit. We found through experiments that increasing the number of positive samples by adding more training datasets improves substantially the generalization capability of the trained classifiers. Increasing the number of negative samples also helps, however, after a certain level, no further improvement is observed; therefore, random sub-sampling of the negative samples with our current setting (10 million) does not deteriorate the performance of the classifiers.

Given a set of positive and negative training samples, we extract 3D Haar wavelet features and train a classifier (e.g., the PBT). After that, we test each voxel in a volume one by one as a hypothesis of the object position using the trained classifier.

As shown in Fig. 2.1a, the classifier assigns each hypothesis a score, and we preserve a small number of candidates (100 in our experiments) with the highest detection score for each volume.

There are various ways to select candidates for the following steps. The first approach is based on the detection score: we keep those candidates with a detection score larger than a threshold. One problem with this approach is that the number of candidates retained varies from volume to volume with a fixed threshold. For a volume with bad image quality, we may find no candidates with a detection score larger than a preset threshold. In many applications, we know there is one and only one target anatomy in a volume. Thus, we select a fixed number of candidates that have the highest detection probability. This approach is more robust and the overall detection speed is consistent from volume to volume.

Sometimes, there may be multiple hypotheses ranking around the cutoff line. For example, if we want to keep 100 hypotheses, we may get five hypotheses with the same score ranking at the 100th. Randomly selecting one from these five hypotheses may introduce randomness in the final detection results, while selecting with a fixed heuristic rule (e.g., selecting the one with the smallest z position) may introduce bias. In our work, we keep all hypotheses ranking around the cutoff line; therefore, the actual number of retained candidates may be slightly larger.

2.2.4 Training of Position-Orientation Estimator

In this step, the task is to jointly estimate the position and orientation. The classification problem is formulated as whether there is an object centered at (X, Y, Z) with orientation (ψ, ϕ, θ) . After object position estimation, we preserve the top 100 candidates, (X_i, Y_i, Z_i) , $i = 1, \dots, 100$. Since we want to estimate both the position and orientation, we need to augment the dimension of candidates. For each position candidate, we quantize the orientation space uniformly to generate hypotheses. In the experiments on heart chamber detection (see Sect. 2.5), the orientation is represented as three Euler angles in the ZXZ convention, ψ , ϕ , and θ . The distribution range of an Euler angle is estimated from the training data. Each Euler angle is quantized within the range using a step size of 0.2 radians (11 degrees). For each position candidate (X_i, Y_i, Z_i) , we augment it with N (about 1,000) hypotheses of orientation, $(X_i, Y_i, Z_i, \psi_j, \phi_j, \theta_j)$, $j = 1, \dots, N$. Some position-orientation hypotheses are close to the ground truth (positive) and others are far away (negative).

The learning goal is to distinguish the positive and negative samples using a trained classifier. Using the normalized distance measure of Eq. (2.6), a hypothesis $(X, Y, Z, \psi, \phi, \theta)$ is regarded as a positive sample if it satisfies both Eqs. (2.7) and

$$\max\{|\psi - \psi^t|, |\phi - \phi^t|, |\theta - \theta^t|\} \leq 0.2, \quad (2.9)$$

where $(\psi^t, \phi^t, \theta^t)$ represent the orientation ground truth. A negative sample has either a large position error, satisfying Eq. (2.8), or a large orientation error,

$$\max\{|\psi - \psi^t|, |\phi - \phi^t|, |\theta - \theta^t|\} > 0.4. \quad (2.10)$$

To capture the orientation information, we have to rotate either the volume or feature templates. We use the steerable features (refer to Sect. 2.3.2), which are efficient to extract under rotation. Similarly, the PBT is used for training and the trained classifier is used to prune the hypotheses to preserve only a few candidates (50 in our experiments).

In the above procedure, the positive/negative training samples of the position-orientation classifier are generated from the augmented position candidates. On some datasets, a couple of good position hypotheses that satisfy Eq. (2.7) may be missed after position estimation. We could add them back to generate more position-orientation hypotheses. However, through comparison experiments, we did not find noticeable improvement in the generalization capability of the trained position-orientation estimator. Therefore, the missed good position candidates are not added back during training. Generally, how the classifier is trained should match how the classifier is used. It is preferable to have the same pose hypothesis generation scheme for both the training and testing procedures since during the testing on an unseen volume, we cannot add missed good position candidates back to generate position-orientation hypotheses.

2.2.5 Training of Position-Orientation-Scale Estimator

The training of the position-orientation-scale estimator is analogous to that of the position-orientation estimator except learning is performed in the full nine dimensional similarity transformation space. The dimension of each retained position-orientation candidate is augmented by searching the scale subspace uniformly and exhaustively. The scale search step is set to two voxels. That means a hypothesis $(X, Y, Z, \psi, \phi, \theta, S_x, S_y, S_z)$ is regarded as a positive sample if, in addition to Eqs. (2.7) and (2.9), it satisfies

$$\max\{|S_x - S_x^t|, |S_y - S_y^t|, |S_z - S_z^t|\} \leq 2 \text{ voxels}, \quad (2.11)$$

where (S_x^t, S_y^t, S_z^t) represent the scale ground truth. A negative sample has a large error in position (Eq. (2.8)), orientation (Eq. (2.10)), or scale

$$\max\{|S_x - S_x^t|, |S_y - S_y^t|, |S_z - S_z^t|\} > 4 \text{ voxels}. \quad (2.12)$$

2.2.6 *Aggregation of Pose Candidates*

The goal of object detection is to obtain a single aggregated estimate of the pose parameters of the object. Multiple detections usually occur around the true object pose since the MSL detectors are insensitive to small changes in the pose parameters. Occasionally, false positives may appear. Some false positives are scattered sparsely, while others may be concentrated around a region that appears similar to the target object.

Intuitively, cluster analysis might help to remove sparsely distributed false positives. However, it is very difficult to perform cluster analysis on a small set of top candidates (e.g., 100) in a nine dimensional pose parameter space. Furthermore, the orientation is represented in a completely different space to the position and scale. To perform clustering we need a distance measurement combining the distances in different marginal spaces. The orientation distance measure needs to be weighted properly to be combined with the position and scale distances. Through experiments, we found that clustering did not improve the accuracy, compared to a simple averaging of the top K ($K = 100$) candidates.

Since each pose candidate has a classification score given by the PBT classifier, we tried a weighted average scheme by assigning a larger weight to a pose candidate with a higher classification score. However, we did not find significant difference to a simple unweighted average.

The robustness of the aggregation with simple averaging is partially explained by the fact that there are far more good pose hypotheses in 3D than 2D. An exponential increase of pose hypotheses for 3D objection detection also comes with a lot of good hypotheses. For example, there are at least $2^9 = 512$ hypotheses within one search step size from the ground truth. If we include hypotheses within an error of two search step sizes, the total number of good hypotheses increases dramatically to $4^9 = 262,144$. By exploiting the rich image information in a 3D volume and the state-of-the-art learning algorithms, our MSL detectors perform quite well. Most of the preserved top 100 pose candidates are good and the small portion of outliers does not affect too much the detection accuracy after averaging.

Note that the average based aggregation only works if we know a priori that there is only a single instance of the object in the volume. If there might be multiple instances of the same object type (e.g., intervertebral disks of the spine), cluster analysis should be used to select multiple final detection results [19, 20]. To avoid the sparse-sample issue in a high dimensional space, clustering is often performed only for the position component of the pose candidates.

2.2.7 *Object Detection in Unseen Volume*

This section provides a summary of the testing procedure on an unseen volume. The input volume is first converted to a low isotropic resolution (e.g., 3 mm) matching the volumes in the training set. All voxels are tested using the trained

position estimator and the top 100 candidates, (X_i, Y_i, Z_i) , $i = 1, \dots, 100$, are kept. Each candidate is augmented with N (about 1,000) hypotheses of orientation, $(X_i, Y_i, Z_i, \psi_j, \phi_j, \theta_j)$, $j = 1, \dots, N$. Next, the trained position-orientation classifier is used to prune these $100 \times N$ hypotheses and the top 50 candidates are retained, $(\hat{X}_i, \hat{Y}_i, \hat{Z}_i, \hat{\psi}_i, \hat{\phi}_i, \hat{\theta}_i)$, $i = 1, \dots, 50$. Similarly, we augment each candidate with M (also about 1,000) hypotheses of scaling and use the trained position-orientation-scale classifier to rank these $50 \times M$ hypotheses. The average of the top K ($K = 100$) candidates is taken as the final aggregated estimate.

In the following, we analyze the computational complexity of heart chamber detection from a cardiac CT volume [34]. For position estimation, all voxels (about 260,000 voxels for a small volume with $64 \times 64 \times 64$ voxels at the 3 mm resolution) are tested for possible object position. There are about 1,000 hypotheses for orientation and scale each. If the parameter space is searched uniformly and exhaustively, there are about 2.6×10^{11} hypotheses to be tested! However, using MSL, we only test about $260,000 + 100 \times 1,000 + 50 \times 1,000 = 4.1 \times 10^5$ hypotheses and reduce the testing by almost six orders of magnitude.

In practice, an irrelevant volume might be fed into the automatic detection and segmentation system due to mis-labeling. For example, the input data to a heart chamber segmentation system may be an abdominal scan without the heart in the volume at all. There are different strategies to handle such situation.

One is the “garbage-in garbage-out” principle, in which the algorithm just tries its best to produce the best segmentation it can achieve. All segmentation results will eventually be double checked by a physician and a non-meaningful segmentation result on a wrong input data can be easily identified and discarded. In the second strategy, the automatic segmentation algorithm is expected to intelligently tell if a target anatomy is present in the data or not. Depending on the presence or absence of the target anatomy, different processing workflows may be invoked later on. In such scenario, a threshold can be set to reject a wrong input. For example, we check the maximum detection score of the preserved position candidates. If it is less than a pre-set threshold, the data is rejected. Normally, we set the threshold very low to avoid rejecting a good input data. Wrong results can later be rejected in the subsequent detection pipelines, e.g., position-orientation estimation and position-orientation-scale estimation.

2.3 3D Image Features

The MSL is an open and flexible object detection framework. Any image features can be integrated into the framework as long as they can provide some information to discriminate between positive and negative hypotheses. In this section, we present two kinds of 3D image features used in a specific implementation of MSL for heart chamber detection in cardiac CT volumes (as presented in Sect. 2.5). The Haar wavelet features are used for object position estimation and the steerable

features, capable of encoding the orientation hypothesis, are used for position-orientation estimation and position-orientation-scale estimation. In fact, these two kinds of features work well on multiple object detection problems in major imaging modalities, including ultrasound data.

2.3.1 3D Haar Wavelet Features

Haar wavelet features were first proposed for object detection by Oren et al. [24]. Figure 2.4a shows a few Haar feature templates, where the value of the feature is calculated by subtracting the sum of pixel intensity inside the dark boxes from the sum of pixel intensity of the bright boxes. Different to the Haar wavelet basis

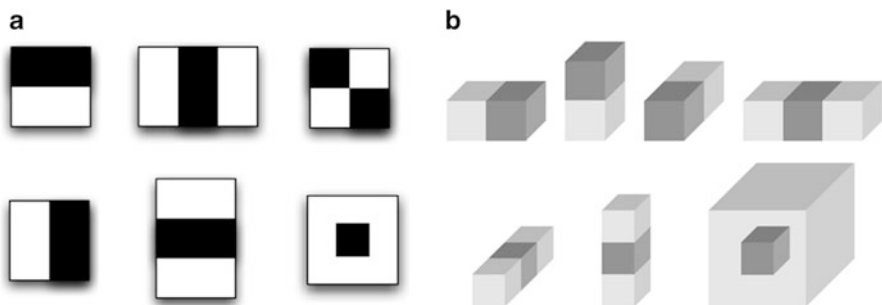


Fig. 2.4 Haar wavelet features are calculated by subtracting the sum of pixel intensity inside the dark boxes from the sum of pixel intensity of the bright boxes. (a) A few 2D Haar feature templates. (b) A few 3D Haar feature templates

functions, Haar features are an open feature family and basically any configuration of the dark and bright boxes can be used, e.g., two-box, three-box, and four-box feature templates as shown in Fig. 2.4a. By translating and scaling the feature templates over the image, we can generate a huge number of image features. Therefore, Haar features are often over-complete and not restricted to the complete linearly-independent Haar wavelet basis functions. Haar features were made popular by [31] with introduction of the integral image, which contributed to a fast computation of Haar features. Using the integral image, the computation of the sum of pixel intensity inside a rectangle at any position and size only involves addition or subtraction of four elements of the integral image (as shown in Fig. 2.5). The computation time is constant no matter how large the rectangle is.

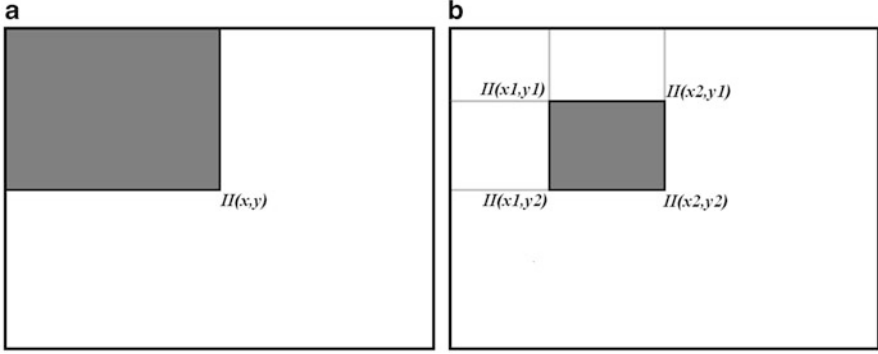


Fig. 2.5 2D integral image for efficient calculation of the sum of intensity of pixels inside a box. (a) Each element $I(x,y)$ of an integral image indicates the sum of intensity of pixels inside the box formed by the origin (the top-left corner) and position (x,y) . (b) Calculating the sum of intensity of any axis-aligned box only involves addition/subtraction of four elements

We reported the first extension of Haar features to 3D in [29]. Figure 2.4b shows a few 3D Haar feature templates. Suppose $I(x,y,z)$ is the intensity of voxel (x,y,z) . The 3D integral image $I(x,y,z)$ is defined as

$$I(x,y,z) = \sum_{u=0}^x \sum_{v=0}^y \sum_{w=0}^z I(u,v,w). \quad (2.13)$$

The sum of voxel intensity inside a box of $x \in [x_1, x_2]$, $y \in [y_1, y_2]$, and $z \in [z_1, z_2]$ is

$$S_{x_1, y_1, z_1}^{x_2, y_2, z_2} = I(x_2, y_2, z_2) - I(x_1, y_2, z_2) - I(x_2, y_1, z_2) - I(x_2, y_2, z_1) + \\ I(x_2, y_1, z_1) + I(x_1, y_2, z_1) + I(x_1, y_1, z_2) - I(x_1, y_1, z_1), \quad (2.14)$$

which involves the addition/subtraction of eight elements of the integral image.

The integral image can be calculated by passing original image once using the algorithm described in [31], as shown in Fig. 2.6. This algorithm can be directly extended to 3D; however, the resulting computation is sub-optimal since it does not benefit from the existence of multiple Central Processing Units (CPU). For example, sequential integral image calculations take about 0.6 s for a brain MRI volume of 1.35 mm resolution with $192 \times 192 \times 149$ voxels. Since the overall MSL-based detection is approaching a sub-second speed, the integral image calculation becomes a computational bottleneck.

Note that almost all up-to-date personal computers have multiple CPUs and therefore it is advantageous to use parallel computing to make full use of all CPUs. Here, we present a 2D integral image calculation method, which is suited for parallel computation (as shown in Fig. 2.7). We pass the image twice to calculate the cumulative sum along rows and columns, respectively. Since each row is processed independently when we calculate the row sums, the computation can be parallelized.

Input: $I(x,y)$ is the intensity of pixel (x,y) .

W is input image width and H is the image height.

Output: The integral image $II(x,y)$.

Algorithm:

Clear the left and top margins of $II(x,y)$ such that $II(-1,:) = 0$ and $II(:, -1) = 0$.

Suppose $s(x,y)$ is the cumulative sum along column (that is y). Clear its top margin $s(:, -1) = 0$.

For $y = 0, 1, \dots, H-1$

 For $x = 0, 1, \dots, W-1$

$s(x,y) = s(x,y-1) + I(x,y)$

$II(x,y) = II(x-1,y) + s(x,y)$

 End

End

Fig. 2.6 A non-parallelized integral image calculation algorithm [31]

Input: $I(x,y)$ is the intensity of pixel (x,y) .

W is input image width and H is the image height.

Output: The integral image $II(x,y)$.

Algorithm:

Clear the left and top margins of $II(x,y)$ such that $II(-1,:) = 0$ and $II(:, -1) = 0$.

/ Calculate the cumulative sum for each row, which can be done with multiple threads independently. */*

For $y = 0, 1, \dots, H-1$ */* This loop can be parallelized. */*

 For $x = 0, 1, \dots, W-1$

$II(x,y) = II(x-1,y) + I(x,y)$

 End

End

/ Calculate the cumulative sum for each column, which can be done with multiple threads independently. */*

For $x = 0, 1, \dots, W-1$ */* This loop can be parallelized. */*

 For $y = 0, 1, \dots, H-1$

$II(x,y) = II(x,y-1) + I(x,y)$

 End

End

Fig. 2.7 An integral image calculation algorithm suitable for parallel computation

The same applies to the calculation of the cumulative sums along columns. Our parallel algorithm does not consume extra memory. Compared to the sequential algorithm we need to pass the image twice; however, the number of mathematical operations (additions) is the same. When the parallel algorithm is implemented on a single CPU, it is as efficient as the sequential algorithm [31]. However, when casted into multi-threading computation, it can make full use of the multiple CPU cores with little overhead in synchronizing multiple threads.

The parallel algorithm can be extended to 3D in a straightforward way. We first calculate the integral image for each slice independently using parallel computation. We then calculate the cumulative sums along the slices (the z axis), which can also be computed independently for each (x,y) . The computation threads only need to

synchronize once to wait for the 2D integral images of all slices to be generated before calculating the cumulative sums along the z direction. On a computer with quad-core CPUs, after using multi-threading, the integral image calculation time is reduced to one third, from 601 to 194 ms, for a brain MRI volume with $192 \times 192 \times 149$ voxels.

2.3.2 Steerable Features

Global features, such as 3D Haar wavelet features, are effective to capture the global information (e.g., scale) of an object. To capture the orientation information of a hypothesis, we should rotate either the volume or the feature templates. However, it is time consuming to rotate a 3D volume and there is no efficient way to rotate the Haar wavelet feature templates. Local features are fast to evaluate but they lose the global information of the whole object.

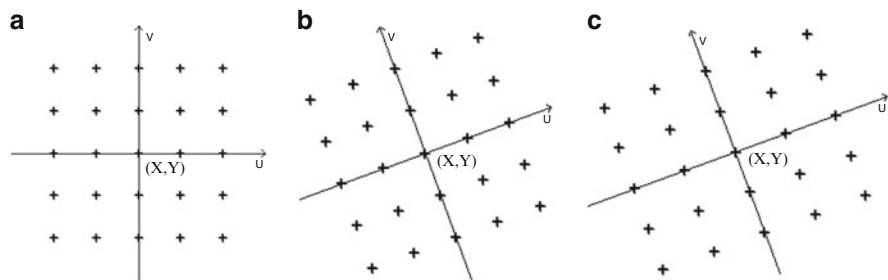


Fig. 2.8 Using a regular sampling pattern to incorporate a hypothesis (X, Y, ψ, S_x, S_y) about a 2D object pose. The sampling points are indicated as '+'. (a) Move the pattern center to (X, Y) . (b) Align the pattern to the orientation ψ . (c) The final aligned sampling pattern after scaling along each axis, proportionally to (S_x, S_y) . ©2008 IEEE. Reprinted, with permission, from Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Four-chamber heart modeling and automatic segmentation for 3D cardiac CT volumes using marginal space learning and steerable features. *IEEE Trans. Medical Imaging* **27**(11), 1668–1681 (2008)

As a solution, we introduced the steerable features, which can capture the global position, orientation, and scale of the object and at the same time can be efficiently computed. To define steerable features, we sample a few points from the volume under a sampling pattern. We then extract a few local features at each sampling point (e.g., voxel intensity and gradient). The novelty is that we embed the global position, orientation, and scale information into the distribution of sampling points, while each individual feature is locally defined. Instead of aligning the volume to the hypothesized orientation, we steer the sampling pattern. This is where the name “steerable features” comes from. In this way, we combine the advantages of both global and local features.

Figure 2.8 shows how to embed a pose hypothesis in steerable features using a regular sampling pattern (illustrated for a 2D case for clearance in visualization). Suppose we want to test if hypothesis $(X, Y, Z, \psi, \phi, \theta, S_x, S_y, S_z)$ is a good estimate of the similarity transformation of the object. A local coordinate system is defined to be centered at position (X, Y, Z) (Fig. 2.8a) and the axes are aligned with the hypothesized orientation (ψ, ϕ, θ) (Fig. 2.8b). A few points (represented as ‘+’ in Fig. 2.8) are uniformly sampled along each coordinate axis inside a rectangular region. The size of the rectangular region along an axis is proportional to the scale of the shape in that direction (S_x , S_y , or S_z) to incorporate the scale information (Fig. 2.8c).

Note that the steerable features are also used for position-orientation estimation and at that stage there are no hypotheses about the scales yet. We use the mean scale values calculated from the training set to resize the sampling pattern. The steerable features constitute a general framework, in which different sampling patterns can be defined. Please refer to Fig. 2.16b for another sampling pattern which can incorporate the nonrigid deformation parameters.

Let us discuss now the local features. At each sampling point, we extract a few local features based on the intensity and gradient from the original volume. A major reason to select these features is that they can be extracted efficiently. Suppose a sampling point (x, y, z) has intensity I and gradient $\mathbf{g} = (g_x, g_y, g_z)$. The three axes of object-oriented local coordinate system are \mathbf{n}_x , \mathbf{n}_y , and \mathbf{n}_z . The angle between the gradient \mathbf{g} and the z axis is $\alpha = \arccos(\mathbf{n}_z \cdot \mathbf{g})$, where $\mathbf{n}_z \cdot \mathbf{g}$ means the inner product between two vectors \mathbf{n}_z and \mathbf{g} . The following 24 features are extracted: I , \sqrt{I} , $\sqrt[3]{I}$, I^2 , I^3 , $\log I$, $\|\mathbf{g}\|$, $\sqrt{\|\mathbf{g}\|}$, $\sqrt[3]{\|\mathbf{g}\|}$, $\|\mathbf{g}\|^2$, $\|\mathbf{g}\|^3$, $\log \|\mathbf{g}\|$, α , $\sqrt{\alpha}$, $\sqrt[3]{\alpha}$, α^2 , α^3 , $\log \alpha$, g_x , g_y , g_z , $\mathbf{n}_x \cdot \mathbf{g}$, $\mathbf{n}_y \cdot \mathbf{g}$, $\mathbf{n}_z \cdot \mathbf{g}$. In total, we have 24 local features at each sampling point.

The first six features are based on intensity and the remaining 18 features are transformations of gradients. Gradients roughly tell us if the sampling point lies on a boundary and they can also be calculated very fast. Feature transformation, a technique often used in pattern classification, is a process through which a new set of features is created [21]. We use it to enhance the feature set by adding a few transformations of an individual feature. Suppose there are P sampling points, we get a feature pool containing $24 \times P$ features. In our case, a $5 \times 5 \times 5$ regular sampling pattern is used for object detection, resulting in $P = 125$ sampling points.

Steerable features can be computed at different levels of an image pyramid and then put together to get a bigger feature pool. For example, a total of $n \times 24 \times P$ features can be extracted on a pyramid with n levels. Computing steerable features in a low resolution volume makes the local features more stable. At a low resolution, the image intensity of a voxel actually corresponds to the average intensity in a small block at a high resolution. In this case, for example, an image intensity feature is less affected by the pepper-and-salt noise and the spectral noise in ultrasound. The gradient can also be calculated more reliably in a smoothed low resolution volume. However, the image pyramid has also limitations. For example, a small object (e.g., a coronary artery) may be smoothed out in a low resolution volume. Furthermore, it may be time consuming to build a pyramid for a large volume. Therefore, it really depends on the application what type of image pyramid is employed, if any. For the

experiments on heart chamber detection (see Sect. 2.5), the steerable features are calculated on a pyramid with three levels (3, 6, and 12 mm, respectively).

By extracting steerable features on a pyramid with three levels, we create a feature pool with a total of $3 \times 24 \times 125 = 9,000$ features. These features are used to train histogram-based weak classifiers [26] and we apply the PBT [28] to combine the selected weak classifiers to achieve a strong classifier. Here are some statistics concerning the selected features by the boosting algorithm, as part of the experiments on heart chamber detection in cardiac CT volumes, as presented in Sect. 2.5. Overall, there are 3,696 features selected by all object detection classifiers. We found that each feature type was selected at least once. The intensity features, I , \sqrt{I} , $\sqrt[3]{I}$, I^2 , I^3 , and $\log I$, counted about 26 % of the selected features, while, the following four gradient-based features, g_x , g_y , g_z , and $\|g\|$, counted about 34 %.

2.4 Classifiers

The MSL is not bounded to a particular classifier and any state-of-the-art learning algorithms can be used to train its classifiers. We employ the Probabilistic Boosting-Tree (PBT) [28] as a default classifier in MSL due to its classification efficiency and capability to deal with unbalanced training samples. The efficiency of the PBT is further improved by adaptively combining it with the classifier cascade [31].

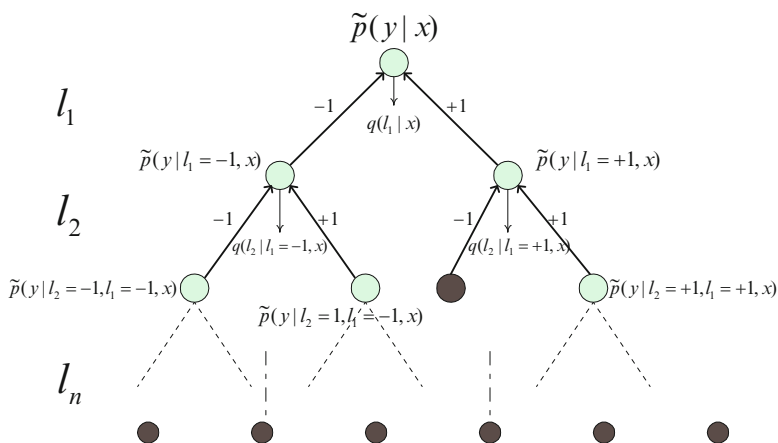


Fig. 2.9 The probabilistic boosting-tree. The dark nodes are the leaf nodes. Each tree node is a strong classifier. Each level of the tree corresponds to an augmented variable l_i . Image courtesy of Zhuowen Tu. ©2005 IEEE. Reprinted, with permission, from Tu, Z.: Probabilistic boosting-tree: Learning discriminative methods for classification, recognition, and clustering. In: Proc. Int'l Conf. Computer Vision, pp. 1589–1596 (2005)

2.4.1 Probabilistic Boosting-Tree

Since a large weak feature pool (in the order of 100,000+ features) can be computed from image data, a process of feature reduction or feature selection is needed for training a classifier. This process is required as a preprocessing step for classifiers such as the Support Vector Machine (SVM) [30], or it is intrinsic to boosting based classifiers [11, 26] and random forests [2]. Classifiers with intrinsic feature selection can train directly on a large feature pool with weak features and this explains the popularity of boosting and random forests. Increased classifier efficiency can be obtained through cascading of classifiers (e.g., AdaBoost [31]) to efficiently detect rare events, e.g., much fewer positive samples than the negatives.

An AdaBoost classifier is an implementation of the boosting technique for a binary classification problem. The final classification decision is based on weighted average of the classification results of the selected weak classifiers

$$H(x) = \sum_{i=1}^K w_i h_i(x), \quad (2.15)$$

where $h_i(x)$ is the classification result (+1 for positive output and -1 for negative output) of a weak classifier; w_i is the weight of each weak classifier, which is determined by the boosting technique. The final output of the class label $C(x)$ is positive if $H(x) > 0$; otherwise, it is negative. An additional threshold θ can be used to tune the performance of an AdaBoost classifier,

$$C(x) = \begin{cases} 1 & \text{if } H(x) > \theta \\ -1 & \text{otherwise} \end{cases}. \quad (2.16)$$

For the detection of rare events, we tune θ to achieve a high sensitivity (which can pass almost all positive samples to the next stage) with a reasonable specificity (e.g., rejecting half of the negative samples); therefore, easy negative samples can be quickly rejected in the early stages of the cascade. A cascade can be viewed as a degenerated decision tree [3, 25].

The Probabilistic Boosting-Tree (PBT) [28] is a tree-structured classifier, that combines the decision tree and AdaBoost classifiers. Besides high accuracy and efficiency, the PBT also outputs an estimate of the posterior distribution, which is helpful to rank the hypotheses. We use the PBT extensively in the MSL based object detection framework. To make this chapter self-contained, we provide a brief introduction of the PBT. An interested reader is referred to [28] for a more detailed description of the algorithm.

As shown in Fig. 2.9, the PBT shares the same tree structure as the decision tree. However, different to the decision tree, each node in the PBT is an AdaBoost strong classifier, instead of a simple weak classifier. Another difference is that the PBT can output an estimate of the posterior distribution $p(y|x)$, instead of just a classification label. Here, x is an observed sample and y is its class label (+1 or -1). In MSL, we

need to rank the hypotheses to preserve a small number of top hypotheses after each MSL classification stage. Therefore, the capability to compute the discriminative probability is very important.

A number of learning algorithms can generate a classification score in addition to the class label. Empirically, the classification score can be normalized to the range $[0, 1]$ to mimic the posterior distribution $p(y|x)$. However, such an ad hoc approach lacks theoretical foundation. In practice, many classifiers are only sensitive to the region around the boundary between positive and negative samples. In other words, the performance is tuned at a specific point (with a normalized classification score of 0.5) on the Receiver-Operating-Characteristic (ROC) curve. Therefore, their performance on other regions is not optimal. From this view, the PBT has an advantage over other learning algorithms such as the decision trees.

Friedman et al. [12] have shown that AdaBoost is approximating logistic regression. The probability computed from an AdaBoost classifier

$$q(+1|x) = \frac{e^{2H(x)}}{1 + e^{2H(x)}}, \quad (2.17)$$

is a good estimate of the posterior probability. Here, $H(x)$ is the weighted sum of the weak classifier responses in AdaBoost, Eq. (2.15). In many real applications, the posterior probability may be very complicated, therefore difficult to estimate accurately. Many weak classifiers (often a few hundred) need to be integrated to approximate the posterior probability well.

The PBT approaches the target posterior probability by data augmentation (tree expansion) through a divide-and-conquer strategy. Figure 2.9 is an illustration of the PBT, where the tree level l_i is an augmented variable. At the top of the tree node, it gathers the information from its descendants and reports an overall posterior probability,

$$\begin{aligned} \tilde{p}(y|x) &= \sum_{l_1} \tilde{p}(y|l_1, x) q(l_1|x) \\ &= \tilde{p}(y|l_1 = 1, x) q(l_1 = 1|x) + \tilde{p}(y|l_1 = -1, x) q(l_1 = -1|x). \end{aligned} \quad (2.18)$$

Here, $q(l_1|x)$ as defined in Eq. (2.17) is the posterior probability estimated by the root AdaBoost classifier. We can expand the posterior probability estimate of a PBT with a depth of $n + 1$ levels as

$$\begin{aligned} \tilde{p}(y|x) &= \sum_{l_1} \tilde{p}(y|l_1, x) q(l_1|x) \\ &= \sum_{l_1, l_2} \tilde{p}(y|l_2, l_1, x) q(l_2|l_1, x) q(l_1|x) \\ &\dots \\ &= \sum_{l_1, \dots, l_n} \tilde{p}(y|l_n, \dots, l_1, x) \dots q(l_2|l_1, x) q(l_1|x). \end{aligned} \quad (2.19)$$

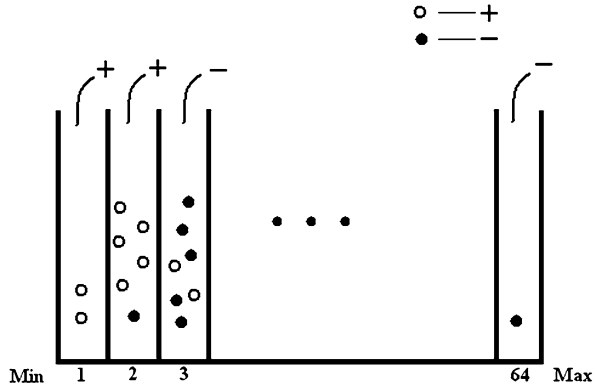
At a tree leaf node, $\tilde{p}(y|l_n, \dots, l_1, x)$ is the posterior probability estimated at that node for class y

$$\tilde{p}(y|l_n, \dots, l_1, x) = \sum_{l_{n+1}} \delta(y = l_{n+1}) q(l_{n+1}|l_n, \dots, l_1, x), \quad (2.20)$$

where $\delta(\cdot)$ is the Dirac delta function.

Different weak classifiers can be used to train the AdaBoost strong classifiers in the PBT. To increase the efficiency, a weak classifier is normally trained on one feature. The decision stump is a very popular weak classifier, which makes a prediction by comparing the value of a single input feature with a threshold [26]. Depending on the polarity, the decision stump may output a positive or negative class label if the feature value is larger than the threshold. Therefore, a decision stump has only two free parameters to train, the polarity (a boolean variable) and the threshold (a real variable). This weak classifier works well to separate unimodal distributions, however, in most typical applications we encounter distributions with multiple modes.

Fig. 2.10 Histogram based weak classifier



To deal with this challenge, the MSL uses by default a histogram based classifier [26], more flexible than a decision stump. Given a weak feature, we calculate its minimum and maximum values from a training set. The distribution range is then uniformly split into a number of bins (64 bins as used throughout our experiments), as shown in Fig. 2.10. In each bin, we count the total weight of positive and negative samples, respectively, that fall inside the bin. Note that the weight of a sample is adjusted by the AdaBoost algorithm at each iteration to assign a higher weight to the previously mis-classified samples. If the total weight of positive samples is larger than that of the negatives in a bin, the bin polarity is set to be positive; otherwise, it is a negative bin. During classification, a sample falling in a positive bin will get a positive class label. Similarly, a sample in a negative bin is classified as negative. The histogram based weak classifier has more parameters (minimum and maximum feature values, polarity of each bin) that can be adapted to

the training set; therefore, it is more powerful than a decision stump. It has capability to separate distributions with multiple modes.

In Sect. 2.3.2, we use feature transformation to enhance the steerable features. For example, for the intensity feature I extracted at a sampling point, we add a few more transformations, including \sqrt{I} , $\sqrt[3]{I}$, I^2 , I^3 , and $\log I$. Such a monotonic feature transformation does not change the classification result of a decision stump. However, it may provide additional classification power to a histogram based weak classifier since a feature after a nonlinear transformation has a different histogram.

As mentioned in Sect. 2.2.3, the distribution of positive and negative samples is skewed. There are far more negative samples (may be up to 10 million in our default setting) than positive samples (often in the order of tens of thousand or less). A classifier should have the capability to handle such a significantly skewed distribution. In the PBT, to train a classification node, we randomly select a small set of positive/negative samples (e.g., 5,000). After training the AdaBoost classifier for a classification node, all available samples at that node are classified. The threshold of the AdaBoost classifier is tuned on all available samples (not the selected subsamples) to achieve the minimum classification error. All samples that are classified as positive (including both true positives and false positives) are passed to the left child of this node and other samples are passed to the right child node. In addition, samples with ambiguity (with a classification score in $[0.4, 0.6]$) are passed to both the left and right children nodes to enrich the training samples of the children nodes.

2.4.2 Combining Classifier Cascade and Tree

Table 2.1 Comparison between probabilistic boosting-tree [28] and classifier cascade [31]

	Probabilistic Boosting-Tree	Cascade
Pros	(1) More powerful for hard classification problems	(1) Efficient for detection of rare events (2) Faster to train (3) Less likely to over-fit
Cons	(1) More likely to over-fit (2) More time-consuming for detection (3) More time-consuming for training	(1) Less powerful for hard problems

The classifier cascade [31] is a widely used structure to combine multiple classifiers to efficiently detect rare events (Fig. 2.11a). In a cascade, a threshold is picked at each classification stage to achieve a perfect or near perfect detection rate for positive samples. Most negative samples can be screened out in the first several stages. However, achieving a near perfect detection rate for positives may cause a high false positive rate, specially when the positives and negatives are hard to separate.

The PBT is more powerful to classify hard samples using a divide-and-conquer strategy of the decision tree [3, 25]. Starting from the tree root, the training samples are split along the tree, level by level. If the tree is very deep, at a leaf node, the number of positive and negative samples is small; therefore, it is almost always possible to separate positive and negative samples using a big weak feature pool. That means, a PBT has potential to over-fit the training data if the tree is fully expanded. In practice, the over-fitting issue is mitigated by limiting the tree depth (e.g., to 5 or 6 levels). Furthermore, a tree node is not trained and further expanded if the number of positive/negative samples falling on this node is small (e.g., less than 100 samples).

In a decision tree, a sample goes from the tree root to a leaf node. The path is determined by the classification result at each node, and the number of classifications is the level of the tree. However, in a PBT, an unknown sample should be classified by all nodes in the tree, and the probabilities given by all nodes are combined to get the final estimate of the posterior probability of a class label, Eq. (2.20). The number of nodes of a PBT is an exponential function of the tree depth. Suppose, a tree has n levels. The number of nodes of a full tree is $2^0 + 2^1 + \dots + 2^{n-1} = 2^n - 1$ (Fig. 2.11b). For comparison, the number of nodes of a cascade with n levels is n . With more nodes, a PBT may consume more training and detection time than a cascade. In the original PBT [28], a heuristic rule is used to reduce the number of probability evaluations. Only samples with an ambiguous classification score (in the range of $[0.4, 0.6]$) are sent to both left and right children nodes for further evaluation. If the classification score is larger than 0.6, the sample is sent only to the left child node by assuming the estimated probability from the right child node is $q(+1|x) = 0$. The same approximation is applied to a sample with sufficient confidence to be negative (classification score less than 0.4). Such an ad hoc solution makes the probability estimate less reliable. Table 2.1 highlights the pros and cons of a PBT and a cascade classifier.

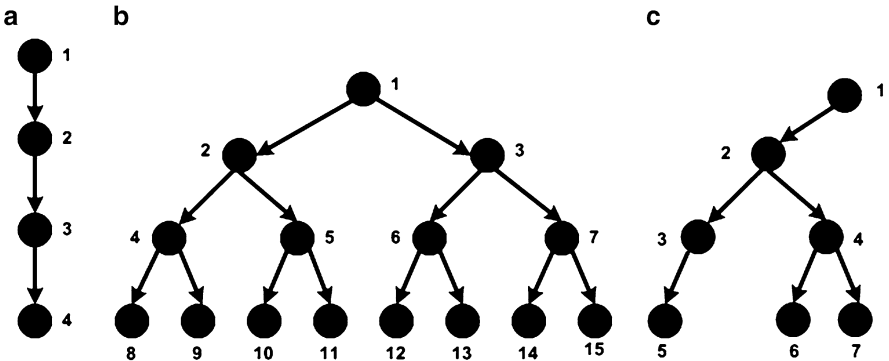


Fig. 2.11 Different structures to combine multiple classifiers. (a) Cascade. (b) Tree. (c) A mixed structure of the cascade and tree

It is possible to combine the advantages of both a cascade and tree. For example, we can put a few levels of cascade before a PBT [29]. The cascade helps to efficiently screen out a large percentage of easy negative samples (e.g., 90 % of negative samples may be filtered by the first two cascades). Since the remaining samples are harder to separate, we can then switch to a tree structure to train more powerful classifiers. However, there are still several issues of this approach. First, we need to manually tune a critical parameter, the number of cascade nodes. If the problem is easy, we should use more cascade nodes to claim the advantage of fast detection speed of the cascade. In addition, for some applications in which a high detection rate is achieved with a high false positive rate, a cascade cannot effectively screen out negatives.

A better approach to combine the advantages of both trees and cascades is to use cascades inside the PBT structure. At each node, we train a strong AdaBoost classifier. After that, we evaluate the trained classifier on all available training samples to this node. If we can achieve a high detection rate (e.g., higher than 97 %) and a low false positive rate (e.g., lower than 50 %), we use the cascade structure. That means we push almost all positive samples to the left child node. The right child node is composed almost purely of negative samples, and it is not necessary to train further. In this way, the structure of the tree is adaptively tuned based on the current training performance. Compared to the naive cascade-before-tree integration, we do not need to manually tune the number of cascade nodes to be appended before a PBT. We only need to set the target detection rate and false positive rate to switch a tree structure to a cascade structure. Figure 2.11c shows a mixed classifier structure with a cascade and tree. During training, after evaluating the classifier node 1, we find its performance can meet the required detection rate and false positive rate. A cascade structure is selected and the right child node of node 1 is not trained. At node 2, we find the classification problem difficult and the trained classifier cannot meet the required accuracy; therefore, a tree structure is used by adding two children nodes to 2. Inside the tree structure, if we find the classification problem becoming easy again, a cascade structure can still be adaptively invoked, e.g., for node 3 in Fig. 2.11c.

2.5 Experiments on Heart Chamber Detection in CT Volumes

In this section, we evaluate MSL on heart chamber detection in cardiac CT volumes. The heart is a complex organ, composed of four chambers, the Left Ventricle (LV), the Right Ventricle (RV), the Left Atrium (LA), and the Right Atrium (RA). We develop a comprehensive four-chamber heart model (refer to Chap. 7), as shown in Fig. 2.12, to provide accurate representation of the anatomies. Our heart model includes the endocardium of all four chambers. In addition, since the epicardium of the left ventricle is clearly visible in a cardiac CT volume, it is also integrated into the heart model.

Under the guidance of cardiologists, we manually annotated all four chambers in 323 cardiac CT volumes (with various cardiac phases) from 137 patients with cardiovascular diseases. Since the LV is clinically more important than other chambers, to improve the system performance on LV detection and segmentation, we annotated extra 134 volumes. In total, we have 457 volumes from 186 patients for the LV. The imaging protocols are heterogeneous with different capture ranges and resolutions. A volume may contain 80–350 slices, while the size of each slice is the same with 512×512 pixels. The resolution inside a slice is isotropic and varies from 0.28 to 0.74 mm for different volumes. The slice thickness (distance between neighboring slices) is larger than the in-slice resolution and varies from 0.4 to 2.0 mm for different volumes.

We use fourfold cross-validation to evaluate the detection algorithm. The whole dataset is randomly split into four roughly equal sets. Three sets are used to train the

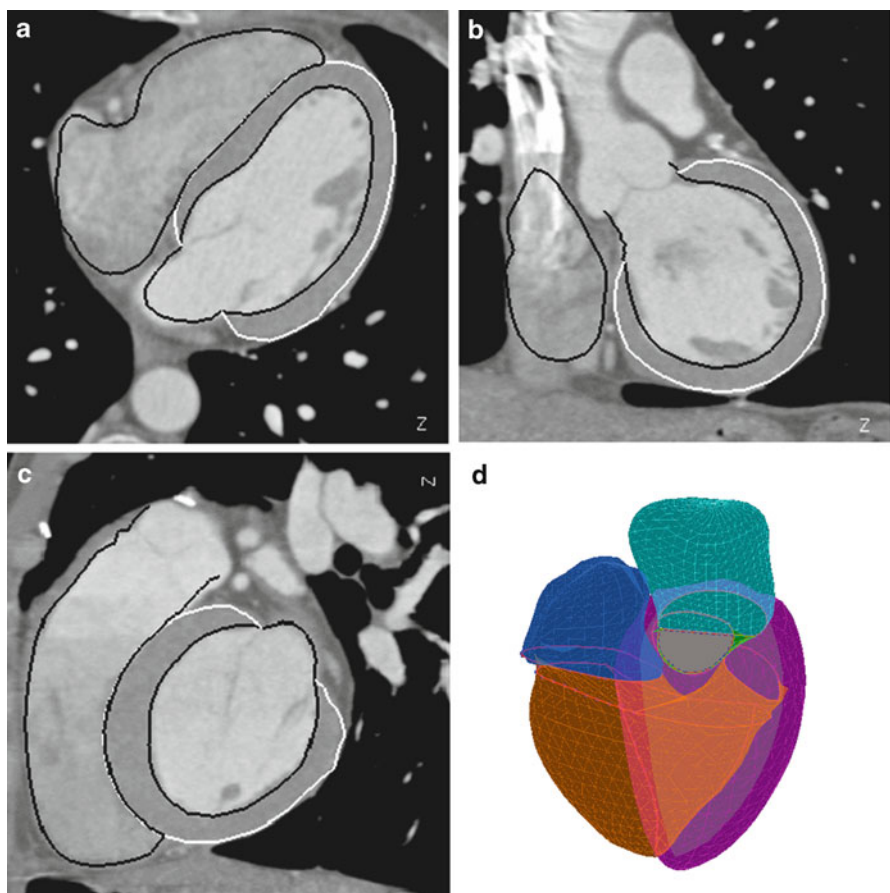


Fig. 2.12 A four-chamber heart model with (a) transaxial view, (b) coronal view, (c) sagittal view, and (d) 3D visualization of the mesh model

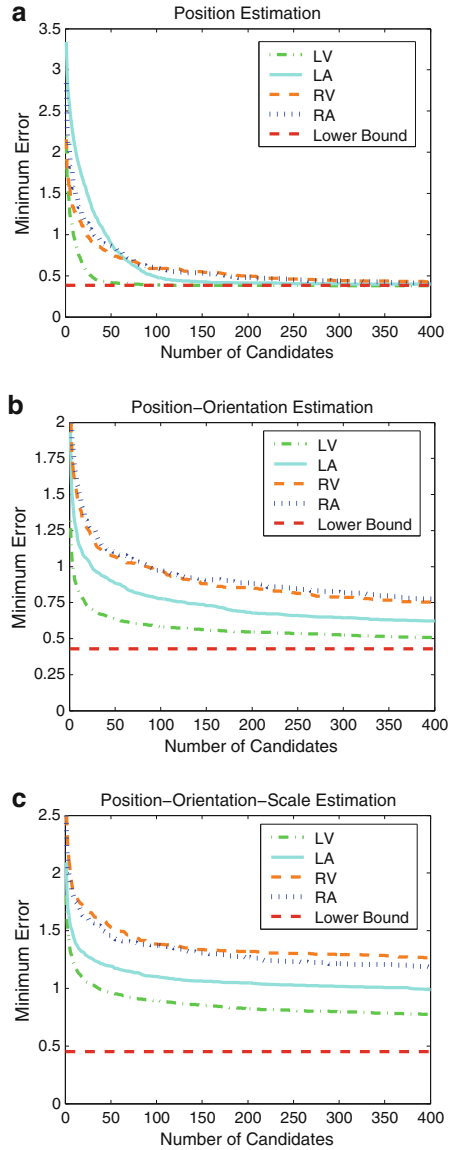
system and the remaining set is reserved for testing. The configuration is rotated, until each set has been tested once. Data from the same patient may have similar shapes and image characteristics since they are often captured on the same CT scanner with the same scanning parameters. If such data appear in both the training and test sets during cross-validation, the result is biased toward a lower detection error. To remove such bias, we enforce the constraint during cross-validation that the volumes from the same patient can only appear in either the training or test set, but not in both.

The search-step-normalized error as defined in Eq. (2.6) is used to evaluate the heart chamber detection accuracy since we can easily distinguish optimal and non-optimal estimates using this error measurement, but not for other error measures such as the weighted Euclidean distance. The optimal estimate is upper-bounded by 0.5 search steps under any search grid. However, a non-optimal estimate has an error larger than 0.5.

The efficiency of MSL comes from the fact that we prune the search space after each step. One concern is that since the space is not fully explored, the MSL might miss the optimal solution at an early stage. In the following, we show that pruning deteriorates only slightly the accuracy in MSL. Figure 2.13 shows the error of the best candidate after each step with respect to the number of candidates preserved. The curves are calculated on all volumes based on cross-validation. The dotted lines at the bottom show the error of the optimal solution under the search grid. As shown in Fig. 2.13a for position estimation, if we keep only one candidate, the average error may be as large as 3.5 voxels. However, by retaining more candidates, the minimum errors have a fast decrease. We have a high probability to keep the optimal solution when 100 candidates are preserved. We observed the same trend in different marginal spaces, such as the position-orientation space as shown in Fig. 2.13b. Based on the trade-off between accuracy and speed, we preserve 50 candidates after position-orientation estimation. After full similarity transformation estimation, the best candidates we get have an error ranging from 1.0 to 1.4 search steps as shown in Fig. 2.13c.

As discussed before, the unweighted averaging of the top candidates into a final estimate achieves the best results. As shown in Fig. 2.14a, the errors decrease quickly with more candidates for averaging until 100 and after that they saturate. Using the average of the top 100 candidates as the final single estimate, we achieve an error of about 1.5–2.0 search steps for different chambers. Figure 2.14b shows the cumulative distribution of errors on all volumes. The LV and LA have smaller errors than the RV and RA since the contrast of the blood pool in the left side of a heart is consistently higher than the right side due to the using of contrast agent (as shown in Figs. 2.12 and 2.15). Compared to the LA, the LV detection error is even smaller because the LV is larger in size and we have more LV training data to cover various cardiovascular diseases. Our approach is robust and we did not observe any major failures. For comparison, the heart detection modules in both [10] and [13] fail on about 10 % volumes. The conclusion of these experiments is that only a small number of candidates are necessary to be preserved after each step, without deteriorating the accuracy of the final estimate.

Fig. 2.13 The searching-step-normalized error, defined in Eq. (2.6), of the best candidate with respect to the number of candidates preserved after each step. **(a)** Position estimation. **(b)** Position-orientation estimation. **(c)** Position-orientation-scale estimation. The dotted lines at bottom show the lower bound of the detection error. ©2008 IEEE. Reprinted, with permission, from Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Four-chamber heart modeling and automatic segmentation for 3D cardiac CT volumes using marginal space learning and steerable features. *IEEE Trans. Medical Imaging* 27(11), 1668–1681 (2008)



We also evaluated the mesh initialization errors after aligning the mean shape with respect to the automatically estimated pose. As a widely used criterion [10, 13, 22], the symmetric point-to-mesh distance, E_{p2m} , is exploited to measure the accuracy in surface boundary delineation. For each point on a mesh, we search for the closest point (not necessarily a mesh triangle vertex) on the other mesh to calculate the minimum Euclidean distance. We calculate the point-to-mesh distance from the detected mesh to the ground truth and vice versa to make the measurement

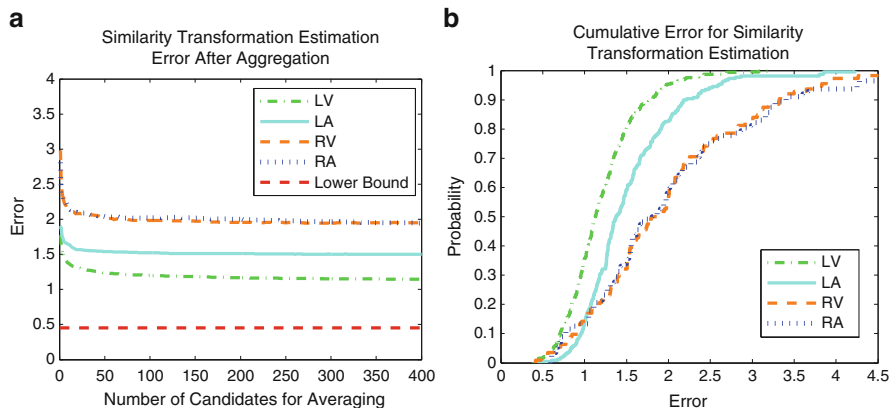


Fig. 2.14 Final similarity transformation estimation error after aggregating multiple candidates using un-weighted average. **(a)** Error vs. the number of candidates for averaging. **(b)** Cumulative distribution of errors on all test datasets using 100 candidates for aggregation

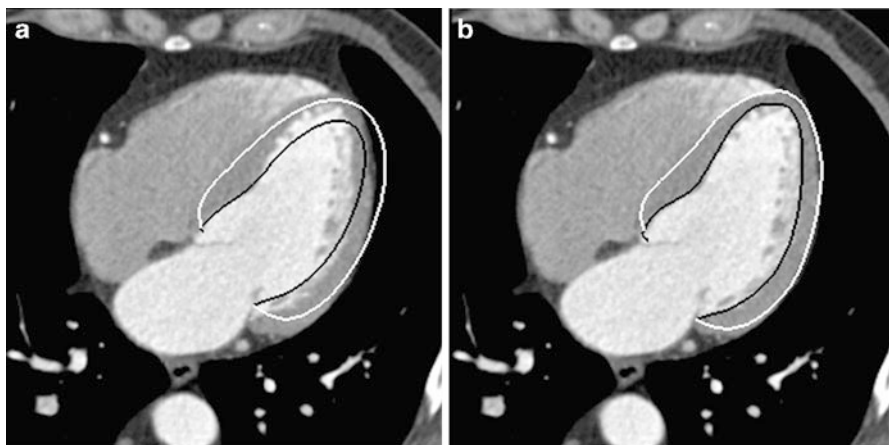


Fig. 2.15 Segmentation of the left ventricle in a cardiac CT volume with a black contour for the endocardium and a white contour for the epicardium. **(a)** Aligned mean shape with respect to the automatically estimated pose. **(b)** Final segmentation

symmetric. In our experiments, we estimate the pose of each chamber separately. The mean E_{p2m} error after heart detection is 3.17 mm for the LV endocardium, 2.51 mm for the LV epicardium, 2.78 mm for the LA, 2.93 mm for the RV, and 3.09 mm for the RA.

Figure 2.15a shows the aligned mean shape of the left ventricle (including both the endocardium and epicardium) with the estimated pose. As we can see, the initial shape is already quite close to the true boundary. After nonrigid deformation estimation of the mesh points, the final segmentation error ranges from 0.84 to

1.57 mm for different chambers. Figure 2.15b shows the final segmentation of the left ventricle. Please refer to Chap. 7 for more details on the nonrigid deformation estimation method.

After code optimization and using the multi-threading techniques, we achieve an average speed of 0.5 s for the automatic detection of a chamber on a computer with a dual-core 3.2 GHz processor and 3 GB memory. Boundary delineation takes about 0.5 s more. If all four chambers are to be segmented, the total computation time is about 4 s.

2.6 Direct Estimation of Nonrigid Deformation Parameters

The standard MSL only estimates the rigid transformation for object localization. In many cases, we want to delineate the boundary of a nonrigid organ. For this purpose, the mean shape is aligned with the estimated object pose as an initial rough estimate of the shape. The Active Shape Model (ASM) [5] is then exploited to deform the initial shape to achieve final boundary delineation. Since the ASM only converges to a local optimum, the initial shape needs to be close to the true object boundary; otherwise, the deformation is likely to get stuck in a suboptimal solution.

This problem is typical for liver segmentation since the liver is the largest organ in human body and is surrounded by several other anatomies: heart, kidney, stomach, and diaphragm. As a soft organ, the liver deforms significantly under the pressure from the neighboring organs. As noted in [16], for a highly deformable shape, the pose estimation can be improved by further initialization. In this section, we present the Nonrigid MSL to directly estimate the nonrigid deformation of an object for better shape initialization.

2.6.1 Nonrigid Marginal Space Learning

There are multiple ways to represent nonrigid deformation. We use the statistical shape model or the so-called Point Distribution Model (PDM) [5] since it can capture the major deformation modes with a few parameters. To build a statistical shape model, we need N shapes, each being represented by M points with anatomical correspondence. By stacking the 3D coordinates of these M points, we get a $3M$ -dimensional vector \mathbf{x} to represent a shape. To remove the relative translation, rotation, and scaling, we first jointly align all shapes using generalized Procrustes analysis (as presented in Sect. 6.3), obtaining the aligned shapes $\mathbf{x}_i, i = 1, 2, \dots, N$. The mean shape $\bar{\mathbf{x}}$ is calculated as the simple average of the aligned shapes, $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$. The shape space spanned by these N aligned shapes can be represented as a linear space with $K = \min\{3M - 1, N - 1\}$ eigen vectors, $\mathbf{V}_1, \dots, \mathbf{V}_K$, based on Principal Component Analysis (PCA) [5]. A new shape \mathbf{y} in the aligned shape space can be represented as

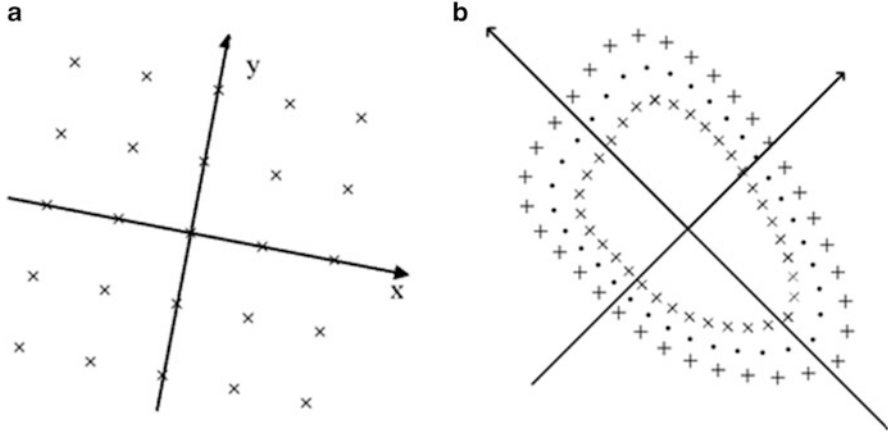


Fig. 2.16 Sampling patterns in steerable features for rigid and nonrigid deformation estimation. (a) Regular sampling pattern for estimating translation, rotation, and scaling. (b) Sampling pattern based on a synthesized shape for estimating nonrigid deformation parameters

$$\mathbf{y} = \bar{\mathbf{x}} + \sum_{j=1}^K c_j \mathbf{V}_j + \mathbf{e}, \quad (2.21)$$

where c_i the so-called PCA coefficient and \mathbf{e} is a $3M$ -dimensional vector for the residual error. Using the statistical shape model, a nonrigid shape can be represented parametrically as $(\mathbf{T}, \mathbf{R}, \mathbf{S}, c_1, \dots, c_K, \bar{\mathbf{x}}, \mathbf{e})$, where $\mathbf{T}, \mathbf{R}, \mathbf{S}$ represent the translation, rotation, and scaling to transfer a nonrigid shape in the aligned shape space back to the world coordinate system.

With this representation, we convert a segmentation (or boundary delineation) problem to a parameter estimation problem. Among all these parameters, $\bar{\mathbf{x}}$ is fixed and \mathbf{e} is sufficiently small if K is large enough (e.g., with enough training shapes). The standard MSL only estimates the rigid part $(\mathbf{T}, \mathbf{R}, \mathbf{S})$ of the transformation. Here, we extend MSL to directly estimate the parameters for nonrigid deformation (c_1, \dots, c_K) .

Given a hypothesis $(\mathbf{T}, \mathbf{R}, \mathbf{S}, c_1, \dots, c_K)$, we train a classifier based on a set of image features F to distinguish a positive hypothesis from a negative one. The image features should be a function of the hypothesis, $F = F(\mathbf{T}, \mathbf{R}, \mathbf{S}, c_1, \dots, c_K)$ and incorporate sufficient information for classification. The steerable features presented in Sect. 2.3.2 of Chap. 2 help to efficiently embed the object pose information into the feature set by steering (translate, rotate, and scale) a sampling pattern with respect to the testing hypothesis. In the original MSL estimation of the rigid transformation, we extract at each sampling point 24 local image features, using a regular sampling pattern to embed the object pose parameters $(\mathbf{T}, \mathbf{R}, \mathbf{S})$. For the Nonrigid MSL, we need to also embed the nonrigid shape parameters $c_j, j = 1, \dots, K$, and achieve this through a new sampling pattern based on the synthesized nonrigid shape (as shown

in Fig. 2.16b). Each hypothesis $(\mathbf{T}, \mathbf{R}, \mathbf{S}, c_1, \dots, c_K)$ corresponds to a nonrigid shape using the statistical shape model (see Eq. (2.21)). We use this synthesized shape as the sampling pattern and extract the local image features on its M points. The dots in Fig. 2.16b are the synthesized shape. If a hypothesis is close to the ground truth, the sampling points should be close to the true object boundary. The image features based on image intensity and local gradient are extracted on these sampling points and help distinguishing the object’s boundary. To capture more information for classification, we also sample a layer of points inside the shape (represented as ‘x’ in Fig. 2.16b) and a layer of points outside the shape (‘+’ in Fig. 2.16b). In total, we get $3M$ sampling points, resulting in a feature pool with $24 \times 3 \times M$ features. Similarly to the standard MSL, we use the boosting technique to learn the classifier.

Due to the exponential increase of testing hypotheses, we cannot train a monolithic classifier to estimate all deformation parameters simultaneously. Using the MSL principle, we split the deformation parameters into groups and estimate them sequentially. To be specific, after position-orientation-scale estimation, we train a classifier in the marginal space of $(\mathbf{T}, \mathbf{R}, \mathbf{S}, c_1, c_2, c_3)$, where (c_1, c_2, c_3) correspond to the top three deformation modes. Given a small set of candidates after position-orientation-scale estimation, we augment them with all possible combinations of (c_1, c_2, c_3) and use the trained nonrigid MSL classifier to select the best hypotheses. In theory, we can apply the MSL principle to estimate more and more nonrigid deformation parameters sequentially. In practice, we find that with the increase of the dimensionality of the marginal spaces, the classifier is more likely to over-fit the data due to the limited number of training samples. No significant improvement has been observed by estimating more than three nonrigid deformation parameters.

2.6.2 Experiments on Liver Detection in 3D CT Volumes

Table 2.2 Comparison of the standard, rigid MSL and Nonrigid MSL on liver detection in 226 CT volumes. Average point-to-mesh error E_{p2m} (in millimeters) of the initialized shape is used for evaluation

	Mean	Std Dev	Median
MSL	7.44	2.26	6.99
Nonrigid MSL	6.65	1.96	6.25

In this experiment, we compare Nonrigid MSL against the standard, rigid MSL on liver detection in 226 3D CT volumes. The dataset is very challenging, including both contrasted and non-contrasted scans, with volumes coming from different clinical sites, generated by different protocols. After object localization, we align the mean shape (a surface mesh) to the estimated transformation. The accuracy of the initial shape estimate is measured with the symmetric point-to-mesh distance E_{p2m} . The initial mesh is then deformed through a subsequent nonrigid estimation step, to fit the image boundary and further reduce the error.

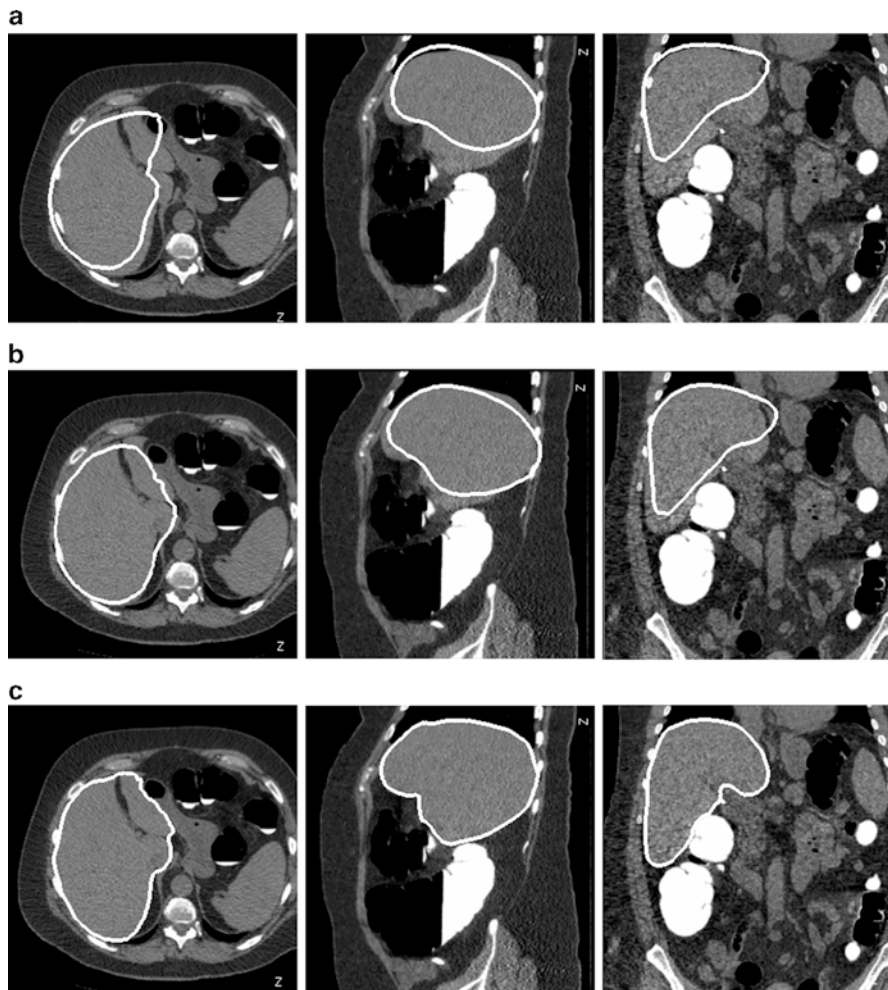


Fig. 2.17 Liver segmentation result on a CT volume. **(a)** Initialization with the mean shape aligned to the translation, orientation, and scale estimated by standard, rigid MSL. **(b)** Initialization by estimating additional three principal components of nonrigid deformation using nonrigid MSL. **(c)** Final segmentation result. From left to right: transaxial, sagittal, and coronal views

Since the focus of this section is to compare the errors corresponding to rigid and nonrigid MSL estimation for object localization, in the following we only measure the error after the MSL estimation. A threefold cross-validation is performed to evaluate the algorithm. As shown in Table 2.2, the rigid MSL achieves a mean initialization error of 7.44 mm and a median error of 6.99 mm after pose estimation. By estimating three more nonrigid deformation parameters (top three PCA coefficients), as shown in the last row of Table 2.2, we can further reduce the average E_{p2m} to 6.65 mm, about 11 % improvement compared to the

original error of 7.44 mm.¹ Figure 2.17a and b show the shape initialization of liver segmentation with baseline MSL and Nonrigid MSL, highlighting that Nonrigid MSL can generate more accurate initial shape. Figure 2.17c shows the final segmentation results obtained by adding the ASM and a learning based boundary detector.

2.7 Theoretical Foundations of Marginal Space Learning

In this section, we provide theoretical justifications of Marginal Space Learning (MSL) and make a connection to the shortest path computation problem in graph search and to particle filtering. The analysis is focused on the standard MSL for estimating the anisotropic similarity transformation parameters of a target object in a 3D volume, although it can be extended to cover Nonrigid MSL.

2.7.1 Relation to Shortest Path Computation

Given an image I , the process of object detection assumes searching for optimal pose parameters, $\hat{\mathbf{T}}$ (translation), $\hat{\mathbf{R}}$ (rotation), $\hat{\mathbf{S}}$ (scaling), to maximize the posterior probability $P(\mathbf{T}, \mathbf{R}, \mathbf{S} | I)$,

$$\hat{\mathbf{T}}, \hat{\mathbf{R}}, \hat{\mathbf{S}} = \arg \max_{\mathbf{T}, \mathbf{R}, \mathbf{S}} P(\mathbf{T}, \mathbf{R}, \mathbf{S} | I). \quad (2.22)$$

In machine learning based object detection approaches, the posterior probability $P(\mathbf{T}, \mathbf{R}, \mathbf{S} | I)$ is estimated using a classifier. As shown in [12], if an AdaBoost classifier is trained to distinguish positive and negative pose hypotheses $(\mathbf{T}, \mathbf{R}, \mathbf{S})$, the probability computed from the AdaBoost classifier (refer to Eq. (2.17)) is a good estimate of the posterior probability.

If the posterior probability is complex, as in most real applications, many weak classifiers, often a few hundred, need to be integrated to approximate well the posterior probability. The PBT classifier as used in our implementation approaches the target posterior probability by data augmentation and tree expansion through a divide-and-conquer strategy. Note that other classifiers, such as the random forests [2] or support vector machine [30], can also provide an approximation of the posterior probability.

To search for an optimal pose of the target object, all possible combinations of the three transformations, \mathbf{T} , \mathbf{R} , and \mathbf{S} , need to be tested by the trained classifier.

¹The reduced error of Nonrigid MSL is achieved by combining the technique of Constrained MSL presented in Chap. 4. Using Constrained MSL, we can reduce the mean E_{p2m} error from 7.44 to 7.12 mm. By estimating three more PCA coefficients of the nonrigid deformation, the mean error is further reduced to 6.65 mm.

Due to the exponential increase of the number of pose hypotheses with respect to the dimensionality of the pose parameter space, exhaustive search is impractical for 3D object detection. In MSL, we split the search into multiple steps. The posterior probability $P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I)$ can be factorized as

$$\begin{aligned} P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I) &= P(\mathbf{T}|I)P(\mathbf{R}, \mathbf{S}|\mathbf{T}, I) \\ &= P(\mathbf{T}|I)P(\mathbf{R}|\mathbf{T}, I)P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I). \end{aligned} \quad (2.23)$$

The position classifier of MSL gives an approximation of the posterior probability $P(\mathbf{T}|I)$. However, in the following estimation steps, we do not estimate the conditional probabilities $P(\mathbf{R}|\mathbf{T}, I)$ and $P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I)$ directly. Instead, the position-orientation classifier estimates probability $P(\mathbf{T}, \mathbf{R}|I)$ and the position-orientation-scale classifier estimates $P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I)$. Nevertheless, the conditional probabilities can be derived as

$$P(\mathbf{R}|\mathbf{T}, I) = \frac{P(\mathbf{T}, \mathbf{R}|I)}{P(\mathbf{T}|I)} \quad (2.24)$$

and

$$P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I) = \frac{P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I)}{P(\mathbf{T}, \mathbf{R}|I)}. \quad (2.25)$$

In MSL, when we evaluate position-orientation hypothesis (\mathbf{T}, \mathbf{R}) to get an estimate of $P(\mathbf{T}, \mathbf{R}|I)$, the probability $P(\mathbf{T}|I)$ is already estimated by the position classifier. So, the conditional probability $P(\mathbf{R}|\mathbf{T}, I)$ can be derived using Eq. (2.24) without an additional evaluation of $P(\mathbf{T}|I)$. The same is true to the calculation of $P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I)$.

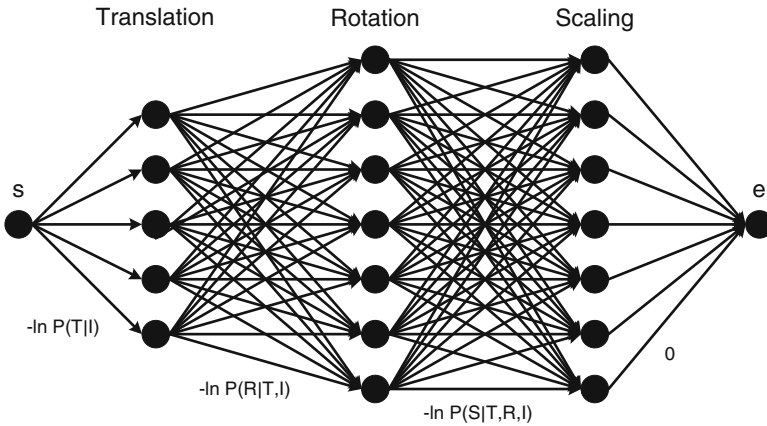


Fig. 2.18 In marginal space learning, object pose estimation can be formulated as searching for the shortest path from the start node s to the end node e in a graph

By factorizing joint probability $P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I)$, the object detection problem as formulated in Eq. (2.22) is converted to

$$\hat{\mathbf{T}}, \hat{\mathbf{R}}, \hat{\mathbf{S}} = \arg \max_{\mathbf{T}, \mathbf{R}, \mathbf{S}} P(\mathbf{T}|I)P(\mathbf{R}|\mathbf{T}, I)P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I). \quad (2.26)$$

In the following, we cast it to the classical shortest path computation problem by defining a path cost function $f(\mathbf{T}, \mathbf{R}, \mathbf{S})$ as

$$\begin{aligned} f(\mathbf{T}, \mathbf{R}, \mathbf{S}) &= -\ln P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I) \\ &= -\ln P(\mathbf{T}|I) - \ln P(\mathbf{R}|\mathbf{T}, I) - \ln P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I). \end{aligned} \quad (2.27)$$

Then, the object pose estimation problem is converted to

$$\hat{\mathbf{T}}, \hat{\mathbf{R}}, \hat{\mathbf{S}} = \arg \min_{\mathbf{T}, \mathbf{R}, \mathbf{S}} -\ln P(\mathbf{T}|I) - \ln P(\mathbf{R}|\mathbf{T}, I) - \ln P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I). \quad (2.28)$$

Figure 2.18 shows the graph of the shortest path computation problem for object pose estimation. The start node s is connected to all position hypotheses \mathbf{T} with a weight $-\ln P(\mathbf{T}|I)$. Each position hypothesis is connected to all orientation hypotheses \mathbf{R} with a weight $-\ln P(\mathbf{R}|\mathbf{T}, I)$. Similarly, each orientation hypothesis is connected to all scale hypotheses \mathbf{S} with a weight $-\ln P(\mathbf{S}|\mathbf{T}, \mathbf{R}, I)$. Finally, each scale hypothesis is connected to the end node e with zero cost. Under this formulation, object pose estimation is equivalent to searching for an optimal path from s to e with a minimum cost.

Exhaustive search guarantees to find a global optimal path by evaluating all potential paths, but it is too time consuming. The A* algorithm [15] is a more efficient search algorithm that also guarantees to find an optimal path if all edge costs are non-negative as ours. Figure 2.19 shows the pseudo code the A* algorithm. It maintains an open list of nodes under evaluation and a closed list of nodes already processed. The nodes in the open list is ordered based on a heuristic function $f(n)$

$$f(n) = g(n) + h(n), \quad (2.29)$$

where $g(n)$ is the cost of the current optimal path from s to n and $h(n)$ is an estimate of the minimum cost from n to e . If $h(n)$ is an under estimate of the real minimum cost, the A* algorithm is guaranteed to find the shortest path. For a graph with non-negative edge costs, $h(n) = 0$ is a trivial heuristic estimate satisfying the under-estimate condition. The A* algorithm is then equivalent to the Dijkstra algorithm [7], another well-known shortest path computation algorithm. However, the more accurate the heuristic estimate $h(n)$ is, the more efficient the search (fewer nodes visited during the search). One limitation of the A* algorithm is that it does not scale well to a large graph. The search with an unbounded open list will eventually fail due to either time or memory constraint.

Create two empty lists: an open list (for nodes under evaluation) and a closed list (for nodes already processed). For each node n in these lists, we record the cost $g(n)$ of the current optimal path from the start node s to n . For each node n in the open list, we also record a heuristic estimate $f(n)$ of the full path cost from s to the end node e passing through n .

Add the start node s ($g(s) = 0$) to the open list

Repeat the following procedure until the open list is empty

 Remove the first element n from the open list

 If n is the end node e

 A shortest path has been found. Terminate the algorithm.

 Else /* n is not the end node */

 Add n to the closed list

 Expand the path to all successor nodes of n .

 For each successor m

 Update cost $g(m) = g(n) + e(n, m)$, where $e(n, m)$ is edge cost from n to m .

 Estimate the cost $h(m)$ of the remaining path from m to e .

 Calculate the heuristic cost $f(m) = g(m) + h(m)$

 If the successor node m is not in the closed list

 Add m to the open list

 Else /* The successor node is in the closed list */

 If $g(m)$ is smaller than its recorded cost in the closed list

 Add m to the open list

 Remove m from the closed list

 End

 End

 End

End

Sort the open list in the ascending order using the estimated cost $f(n) = g(n) + h(n)$

End

Fig. 2.19 Pseudo code of the A* search algorithm [15]

In many real applications, searching for the exact global shortest path is not absolutely necessary. An efficient algorithm is more desirable if it can find a near-optimal path much faster than the exact algorithms. Object detection is such an application. First, there are many good pose hypotheses. For 3D object detection, in total, there are at least $2^9 = 512$ pose hypotheses within one searching step distance to the ground truth. Any of them is good enough. Second, the PBT classification score is just an approximation of the posterior probabilities $P(\mathbf{T}|I)$, $P(\mathbf{T}, \mathbf{R}|I)$, and $P(\mathbf{T}, \mathbf{R}, \mathbf{S}|I)$. That means the edge cost is a noisy measurement. Although the pose hypothesis corresponding to the global optimal path is generally a good estimate, it may not be the one closest to the ground truth.

Many greedy search algorithms have been proposed to reduce the searching time and memory footprint of exact search algorithms. Beam search is a greedy search that was first used in the Harpy Speech Recognition System developed at Carnegie Mellon University in mid-1970s [23]. The basic idea of beam search is that instead of maintaining all paths currently under evaluation, it only maintains a limited number of most promising paths [1]. The upper limit of the number of

maintained paths is called beam width, which is a critical parameter to control the trade-off between search speed (as well as memory footprint) and optimality of the generated path. If the beam width is infinity, the algorithm is equivalent to an exact search that can generate a global optimal path, but at the cost of longer search time and larger memory footprint. With a small beam width, the algorithm is efficient but often generates a path with a large cost or fails to generate a solution at all since an optimal (or feasible) path may be pruned erroneously during the search. To the extreme, with a beam width of one, the algorithm is equivalent to the hill-climbing algorithm, which often outputs a locally optimal solution of poor quality [32].

Although the idea is simple, beam search works well on a lot of problems, e.g., planning [35], scheduling [6], speech recognition [23], and machine translation [27]. It works especially well on problems with a lot of near-optimal solutions [6] like our object pose estimation problem. A recent comparison [32] on a variety of graph search problems shows that beam search offers a better trade-off between speed and accuracy than other greedy search algorithms. It is well suited for large search problems when other algorithms fail to generate a solution at all. In addition, beam search can be integrated into different exact search algorithms, e.g., the A* algorithm.

Note that a tree can be searched in different orders, e.g., depth-first, breadth-first, or best-first, where the nodes are ordered using a heuristic cost estimate. From this point of view, the MSL uses the classical breadth-first beam search. Our graph is basically a tree if the end node e is removed. (The node e is added to convert the graph search problem to a shortest path computation problem. The tree leaf nodes are connected to e with edges of a zero cost.) In the breadth-first search order, a tree is searched level by level. First, the root node is searched; then, all children of the root node are searched; and next, all grand-children of the root node are searched. This search order is propagated level by level until all leaf nodes have been searched.

In MSL, after each step, a limited number of pose hypotheses are preserved and all other pose hypotheses with a low classification score are discarded. The beam width (i.e., the number of preserved candidates after each step) is tuned to get a good trade-off between the estimation accuracy and speed. As shown in Fig. 2.13 for heart chamber detection, a beam width of one (equivalent to the hill-climbing algorithm) is too restrictive and good pose candidates are likely to be pruned erroneously. When the beam width is increased to 100, the best position hypotheses (which are closest to the ground truth) are almost sure to be preserved for the left ventricle detection, as shown in Fig. 2.13a. For the following position-orientation and position-orientation-scale estimation, with a small beam width, we start to lose some of the best pose candidates. Fortunately, our problem contains a lot of near-optimal solutions, and the MSL successfully preserves many good pose candidates, as shown in Fig. 2.13b and c.

2.7.2 *Relation to Particle Filtering*

The MSL can also be regarded as a special type of particle filter [9] for which the particles are propagated in the sequence of subspaces (Eq.(2.5)) of increasingly larger dimension, namely, the marginal spaces. A typical particle filter propagates particles in the same space; hence, the MSL contributes to an extended filtering process, by allowing the space dimensions to vary between estimation steps. For a given marginal space, the MSL prunes the particles through classifiers trained in that subspace, while the propagation to the next subspace involves adding more parameters, thus increasing the particle dimensionality, and adding more particles to cover the new subspace. The last space in the estimation is the pose parameter space of the object that needs to be detected.

2.8 Conclusions

In this chapter, we discussed in detail the Marginal Space Learning (MSL) paradigm, demonstrating its efficiency and robustness. By progressively learning in subspaces of increasing dimensions called Marginal Spaces, the MSL achieves early pruning of irrelevant hypotheses. This property makes MSL an ideal tool for fast pose estimation. The chapter also covered the entire framework of an MSL application in 3D with a focus on heart chamber detection in CT volumes, analyzing its 3D image features and a default classifier, the PBT. Finally we presented an extension to Nonrigid MSL and related the MSL concept to prior search and estimation strategies.

References

1. Bisiani, R.: Beam search. In: S.C. Shapiro (ed.) *Encyclopedia of Artificial Intelligence*, pp. 56–58. John Wiley & Sons (1987)
2. Breiman, L.: Random forests. *Machine Learning* **45**(1), 5–32 (2001)
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: *Classification and regression trees*. Chapman and Hall/CRC (1984)
4. Cootes, T.F., Edwards, G.J., Taylor, C.J.: Active appearance models. *IEEE Trans. Pattern Anal. Machine Intell.* **23**(6), 681–685 (2001)
5. Cootes, T.F., Taylor, C.J., Cooper, D.H., Graham, J.: Active shape models—their training and application. *Computer Vision and Image Understanding* **61**(1), 38–59 (1995)
6. Dashti, M.T., Wijs, A.J.: Pruning state spaces with extended beam search. In: *Proc. Int’l Conf. Automated Technology for Verification and Analysis*, pp. 543–552 (2007)
7. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik* **1**(1), 269–271 (1959)
8. Dollár, P., Tu, Z., Belongie, S.: Supervised learning of edges and object boundaries. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 1964–1971 (2006)

9. Doucet, A., Johansen, A.: A tutorial on particle filtering and smoothing: Fifteen years later. Technical Report, Department of Statistics, University of British Columbia (2008)
10. Ecabert, O., Peters, J., Weese, J.: Modeling shape variability for full heart segmentation in cardiac computed-tomography images. In: Proc. of SPIE Medical Imaging, pp. 1199–1210 (2006)
11. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences* **55**(1), 119–139 (1997)
12. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: A statistical view of boosting. *The Annals of Statistics* **28**(2), 337–407 (2000)
13. Fritz, D., Rinck, D., Unterhinninghofen, R., Dillmann, R., Scheuring, M.: Automatic segmentation of the left ventricle and computation of diagnostic parameters using regiongrowing and a statistical model. In: Proc. of SPIE Medical Imaging, pp. 1844–1854 (2005)
14. Georgescu, B., Zhou, X.S., Comaniciu, D., Gupta, A.: Database-guided segmentation of anatomical structures with complex appearance. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 429–436 (2005)
15. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics* **4**(2), 100–107 (1968)
16. Heimann, T., Münzing, S., Meinzer, H.P., Wolf, I.: A shape-guided deformable model with evolutionary algorithm initialization for 3D soft tissue segmentation. In: Proc. Information Processing in Medical Imaging, pp. 1–12 (2007)
17. Henry, W.L., DeMaria, A., Gramiak, R., King, D.L., Kisslo, J.A., Popp, R.L., Sahn, D.J., Schiller, N.B., Tajik, A., Teichholz, L.E., Weyman, A.E.: Report of the American Society of Echocardiography Committee on Nomenclature and Standards in two-dimensional echocardiography. *Circulation* **62**(2), 212–215 (1980)
18. Karney, C.F.F.: Quaternions in molecular modeling. *Journal of Molecular Graphics and Modeling* **25**(5), 595–604 (2007)
19. Kelm, B.M., Wels, M., Zhou, S.K., Seifert, S., Suehling, M., Zheng, Y., Comaniciu, D.: Spine detection in CT and MR using iterated marginal space learning. *Medical Image Analysis* **17**(8), 1283–1292 (2013)
20. Kelm, B.M., Zhou, S.K., Suehling, M., Zheng, Y., Wels, M., Comaniciu, D.: Detection of 3D spinal geometry using iterated marginal space learning. In: Proc. MICCAI Workshop Medical Computer Vision — Recognition Techniques and Applications in Medical Imaging, pp. 96–105 (2010)
21. Kusiak, A.: Feature transformation methods in data mining. *IEEE Trans. Electronics Packaging Manufacturing* **24**(3), 214–221 (2001)
22. Lorenz, C., von Berg, J.: A comprehensive shape model of the heart. *Medical Image Analysis* **10**(4), 657–670 (2006)
23. Lowerre, B.: The Harpy speech recognition system. Ph.D. thesis (1976)
24. Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., Poggio, T.: Pedestrian detection using wavelet templates. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 193–199 (1997)
25. Quinlan, J.R.: Induction of decision trees. *Machine Learning* **1**(1), 81–106 (1986)
26. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. *Machine Learning* **37**(3), 297–336 (1999)
27. Tillmann, C., Ney, H.: Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics* **29**(1), 97–133 (2003)
28. Tu, Z.: Probabilistic boosting-tree: Learning discriminative methods for classification, recognition, and clustering. In: Proc. Int’l Conf. Computer Vision, pp. 1589–1596 (2005)
29. Tu, Z., Zhou, X.S., Barbu, A., Bogoni, L., Comaniciu, D.: Probabilistic 3D polyp detection in CT images: The role of sample alignment. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 1544–1551 (2006)
30. Vapnik, V.: *The Nature of Statistical Learning Theory*. Springer-Verlag (1995)
31. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 511–518 (2001)

32. Wilt, C., Thayer, J., Ruml, W.: A comparison of greedy search algorithms. In: Proc. Symposium on Combinatorial Search, pp. 1–8 (2010)
33. Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Fast automatic heart chamber segmentation from 3D CT data using marginal space learning and steerable features. In: Proc. Int'l Conf. Computer Vision, pp. 1–8 (2007)
34. Zheng, Y., Barbu, A., Georgescu, B., Scheuering, M., Comaniciu, D.: Four-chamber heart modeling and automatic segmentation for 3D cardiac CT volumes using marginal space learning and steerable features. *IEEE Trans. Medical Imaging* **27**(11), 1668–1681 (2008)
35. Zhou, R., Hansen, E.A.: Beam-stack search: Integrating backtracking with beam search. In: Int'l Conf. on Automated Planning and Scheduling, pp. 90–98 (2005)

Marginal Space Learning for Medical Image Analysis
Efficient Detection and Segmentation of Anatomical
Structures

Zheng, Y.; Comaniciu, D.

2014, XX, 268 p. 122 illus., 58 illus. in color., Hardcover

ISBN: 978-1-4939-0599-7