

Chapter 7

Real Time Hand Pose Estimation Using Depth Sensors

Cem Keskin, Furkan Kıraç, Yunus Emre Kara, and Lale Akarun

Abstract Real-time hand posture capture has been a difficult goal in computer vision. The extraction of hand skeleton parameters would be an important milestone for sign language recognition, since it would make classification of hand shapes and gestures possible. The recent introduction of the Kinect depth sensor has accelerated research in human body pose capture. This chapter describes a real-time hand pose estimation method employing an object recognition by parts approach, and the use of this method for hand shape classification. First, a realistic 3D hand model is used to represent the hand with 21 different parts. Then, a random decision forest (RDF) is trained on synthetic depth images generated by animating the hand model, which is used to perform per pixel classification and to assign each pixel to a hand part. The classification results are fed into a local mode finding algorithm to estimate the joint locations for the hand skeleton. The system can process depth images retrieved from Kinect in real time, and does not rely on temporal information. As a simple application of the system, we also describe a support vector machine (SVM)-based recognition module for the ten digits of American Sign Language (ASL) based on our method, which attains a recognition rate of 99.9 % on live depth images in real time.

7.1 Introduction

After the release of multi-touch enabled smart phones and operating systems, there has been a renewed interest in natural interfaces and particularly in hand gestures.

C. Keskin (✉) · F. Kıraç · Y.E. Kara · L. Akarun
Computer Engineering Department, Boğaziçi University, 34342, Istanbul, Turkey
e-mail: keskinc@cmpe.boun.edu.tr

F. Kıraç
e-mail: kiracmus@boun.edu.tr

Y.E. Kara
e-mail: yunus.kara@boun.edu.tr

L. Akarun
e-mail: akarun@boun.edu.tr

Hand gestures are used in these systems to interact with programs such as games, browsers, e-mail readers and a diverse set of tools. Immersive 3D displays will also depend heavily on the use of hand gestures for interaction.

Vision-based hand gesture recognition, and particularly, sign language recognition has attracted the interest of researchers for more than 20 years. Yet, a framework that robustly detects the naked hand and recognizes hand poses and gestures from color images has continued to be elusive. This can be attributed mostly to the large variance of the retrieved images, caused by changing light conditions, and to the difficulty of distinguishing the hand from other body parts.

Two developments have recently accelerated implementations of human–computer interaction using human body and hand gestures: The first is the release and widespread acceptance of the Kinect depth sensor [1]. With its ability to generate depth images in very low illumination conditions, this sensor makes the human body and hand detection and segmentation a simple task. The second development is the use of fast discriminative approaches using simple depth features coupled with GPU implementation; enabling real-time human body pose extraction [2, 3].

Recently, Kinect has been used to achieve real-time body tracking capabilities, which has triggered a new era of natural interface-based applications. In their revolutionary work, Shotton et al. fit a skeleton to the human body using their object recognition-based approach [2]. They use a large amount of labeled real and synthetic images to train a randomized decision forest (RDF) [20] for the task of body part recognition. In a later study, Girschick et al. [3] use the same methodology with a regression forest, and let each pixel vote for joint coordinates. Detailed explanation of both frameworks can be found in the first chapter of this part of the book.

The object recognition by parts approach is applicable to the hand pose estimation problem as well, but there are some notable differences between the human body and hand: (i) The projected depth image of a hand is much smaller than that of a body; (ii) a body can be assumed to be upright but a hand can take any orientation; (iii) in the case of hands, the number of possible meaningful configurations is much higher and the problem of self-occlusion is severe. On the other hand, the inter-personal variance of the shape of hands is much smaller compared to the huge differences between fully clothed human bodies.

In this work, we largely follow the approach in [2]. Adopting the idea of an intermediate representation for the object whose pose is to be estimated, we generate synthetic hand images and label their parts, such that each skeleton joint is at the center of one of the labeled parts. We form large datasets created from random and manually set skeleton parameters, and train several randomized decision trees (RDT) [20], which are then used to classify each pixel of the retrieved depth image. Finally, we apply the mean shift algorithm to estimate the joint centers as in [2]. The resulting framework can estimate hand poses in real time.

As a proof of concept, we demonstrate the system by using it to recognize ASL digits. In our approach, we first train an RDF for ASL digits using synthetic images. Then, we evaluate real depth images of ASL and fit a skeleton to each image. Finally, we classify the skeleton configuration parameters using SVM, which can then be used to infer the hand shape class for a given real hand depth image, by fitting

a skeleton to it first. We demonstrate that this technique achieves a 99.9 % test accuracy on a dataset of size 15k, collected from five users. Note that there are other simpler approaches to classify ASL digits. However, using the skeletal configuration to classify hand shapes is a powerful method, since the appearance of the hand is entirely determined by the skeleton. This is reflected in the high success rate the framework achieves. More importantly, this approach does not rely on class- or application-specific heuristics and is directly applicable to all types of hand shape.

7.1.1 Related Work

With the release of Kinect, libraries for basic hand gesture recognition tasks have been developed. However, these only consider hand movement, and not hand pose. The estimation of the hand skeleton configuration has largely remained unsolved.

7.1.1.1 Hand Pose Estimation

Most approaches to hand pose estimation problem make use of regular RGB cameras. Erol et al. [4] divide the pose estimation methods into two main groups in their review: partial and full pose estimation methods. They further divide the full pose estimation methods into single frame pose estimation and model-based tracking methods. Athitsos et al. [5] estimate 3D hand pose from a cluttered image. They create a large database of synthetic hand poses using an articulated model and find the closest match from this database. Similarly, Romero et al. [6] propose a non-parametric, nearest neighbor-based search in a large database to estimate articulated hand poses. De Campos and Murray [7] use a relevance vector machine-based learning method [8] for single frame hand pose recovery. They combine multiple views to overcome the self-occlusion problem. They also report single and multiple view performances for both synthetic and real images. Rosales et al. [9] use monocular color sequences for recovering 3D hand poses. Their system maps image features to 3D hand poses using specialized mappings. Stergiopoulou and Papamarkos [10] fit a neural network into the detected hand region. They recognize the hand gesture using the grid of the produced neurons. De La Gorce et al. [11] use model-based tracking of the hand pose in monocular videos. Stenger et al. [13] apply model-based tracking using an articulated hand model and estimate the pose with an unscented Kalman filter. Bray et al. [14] propose an algorithm that wraps a particle filter around multiple stochastic meta-descent-based trackers to form a smart particle filter that can track an articulated hand pose. However, the resulting framework does not run in real time. Heap et al. [15] describe a 3D deformable point distribution model of the hand, which is used to track hands using a single RGB camera.

A number of approaches have been reported to estimate the hand pose from depth images. Mo and Neumann [16] use a laser-based camera to produce low-resolution

depth images. They interpolate hand pose using basic sets of finger poses and inter-relations. Malassiotis and Strintzis [17] extract PCA features from depth images of synthetic 3D hand models for training.

In a recent study Oikonomidis et al. [12] present a solution that makes use of both depth and color images. They propose a generative single hypothesis model-based pose estimation method. They use particle swarm optimization for solving the 3D hand pose recovery problem, and report accurate and robust tracking in near real time (15 fps), with a GPU-based implementation.

7.1.1.2 Hand Shape Recognition from Depth

Uebersax et al. propose a system that segments the hand and estimates the hand orientation from captured depth data. Their letter classification method is based on average neighborhood margin maximization. Liu and Fujimura [18] recognize hand gestures using depth images acquired by a time-of-flight camera. The authors detect hands by thresholding the depth data and use Chamfer distance to measure shape similarity. Then, they analyze the trajectory of the hand and classify gestures using shape, location, trajectory, orientation and speed features. Suryanarayan et al. [19] use depth information and recognize scale and rotation invariant poses dynamically. They classify six signature hand poses using a volumetric shape descriptor which they form by augmenting 2D image data with depth information. They use SVM for classification. Uebersax et al. [21] provide a thorough review of ASL letter recognition on depth data.

In Sect. 7.2 we describe the methodology used for fitting the skeleton. Section 7.3 lists the details of conducted experiments and presents our results on ASL digit recognition. Finally, we conclude the chapter in Sect. 7.4.

7.2 Methodology

The flowchart of the system can be viewed in Fig. 7.1. The training phase is given in the upper row, and the evaluation phase in the lower row. As there is no practical way of labeling real depth images, only synthetic images are used for training. To this end, the system employs a realistic 3D hand model that can be configured to form new poses. Our framework handles automatic generation and labeling of synthetic training images by manually setting or randomizing each skeleton parameter. It can then form large datasets by interpolating poses and perturbing joints via addition of Gaussian noise to each joint angle without violating skeleton constraints. These synthetic datasets contain depth-label image pairs for each configuration. Typically, datasets consisting of 40k to 200k image pairs are generated, which are used to train the models. Each tree learns to map the pixels in a depth image to their corresponding labels in the ground truth image. Multiple decision trees are trained that form small ensembles, i.e. forests.

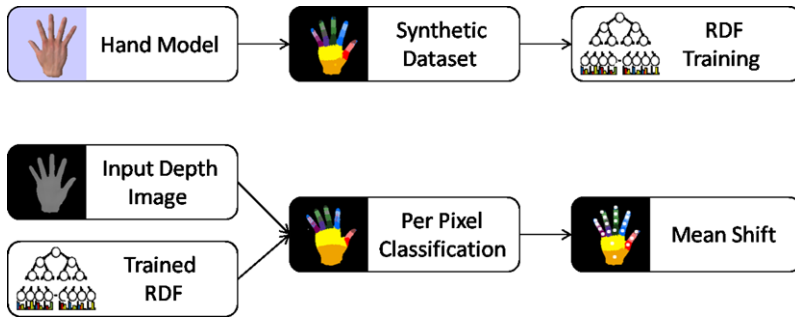


Fig. 7.1 Flowchart of the system. The *top image* depicts the training phase, and the *bottom image* depicts the evaluation phase

In the evaluation phase, an input depth image is fed into the trained RDF without the ground truth labels. The trees in the RDF classify each depth pixel into a hand part by assigning a set of posterior probabilities to it. The posterior probabilities of each tree are averaged over all the trees in the forest. Finally, the mean shift algorithm is used to estimate the 3D coordinates of the centroids of each hand part. The skeleton is formed by connecting the joint coordinates according to the model hierarchy.

The overall accuracy of the system depends on a variety of factors, such as the number of trees, the depth of individual trees, the degree of variation in the training set and other training parameters. In particular, if the training images do not reflect the variety of hand poses encountered in real life, the trees cannot generalize well to unseen poses.

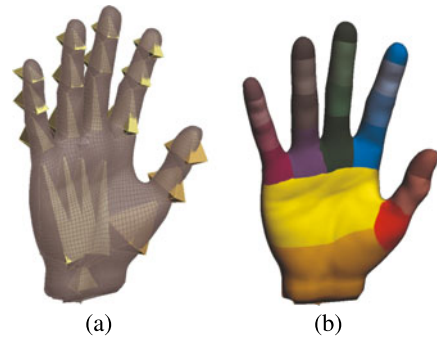
By synthesizing training images, it is possible to automatically create a very large set of configurations. First, a smaller set of plausible and common hand poses is manually created, from which new poses are generated by extrapolating and randomizing these configurations.

7.2.1 Data

To generate the synthetic images, we use a 3D skinned mesh model with a hierarchical skeleton, consisting of 19 bones, 15 joints and 21 different parts as viewed in Fig. 7.2. Hand parts are defined such that all significant skeleton joints are located near the centroids of corresponding parts. Hence, the thumb contains three parts and all the other fingers contain four parts that signify each bone tip. The palm is divided into two different parts, so that the deformations are better captured.

The model is animated and rendered with the texture depicting the hand parts, without shading. The final color image is the label image, and the content of the Z-buffer, which contains the depth map of the rendered pixels, is the depth image. The magnitudes of the depth pixels are mapped to the interval $[0, 255]$ to minimize

Fig. 7.2 The 3D hand model with a hierarchical skeleton and 21 labeled parts that is used to generate a synthetic training set. (a) The skeleton is depicted with *yellow parts* indicating the joint locations. Image (b) shows the parts, each of which correspond to a joint or bone tip in the skeleton



memory cost. To retain compatibility with the incoming Kinect data, the input depth images are also normalized to the same interval.

The training sets are designed with target applications in mind, so that the trained trees can generalize well to previously unseen hand poses that can be encountered during common tasks, such as hand poses used for games, natural interfaces and sign languages, all of which are manually modeled using a tool. The animator tool can interpolate between these poses using the hierarchical skeleton model, and add slight variations to each frame by perturbing joint locations, while changing the camera pose. Skeletal constraints are applied to each interpolated pose, ensuring that the resulting configurations are feasible. A data glove, which measures the joint angles of the hand in real time, can also be used to manipulate the digital model and create realistic hand poses. It can also be used to estimate and better model the inter-personal variations in hand shape, such as size, finger lengths and thickness. However, the models trained on a synthetic dataset formed by manipulating a single hand shape has been found to be sufficient for all types of hand, as inter-personal variance is low for the hands, and the trained models can easily be adapted to different hand sizes by scaling feature parameters if necessary.

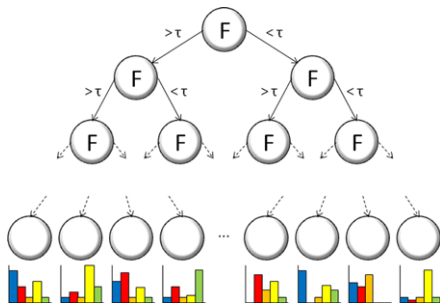
7.2.2 Decision Trees

Decision trees consist of split nodes, which are the internal nodes used to test the input; and leaf nodes, which are the terminal nodes used to infer a set of posterior probabilities for the input, based on statistics collected from training data. Each split node sends the incoming input to one of its children, according to the test result. The test associated with a split node is usually of the form

$$f_n(F_n) < T_n \quad (7.1)$$

where $f_n(F_n)$ is a function of a subset of features and T_n is a threshold, at split node n . The input is injected at the root node, which is forwarded by the split nodes according to the test results; and the posterior probabilities associated with the leaf node that is reached are used to infer the class label. Hence, the training of a deci-

Fig. 7.3 A decision tree. The input pixels are tested at each node and guided *down the tree*, finally reaching a leaf node that is associated with a set of posterior probabilities, which is estimated from the label histogram of data collected during the training



sion tree involves determining the tests and collecting statistics from a training set. A decision tree is depicted in Fig. 7.3.

In the case of an RDT, the features do not need to be determined beforehand. Instead, the feature parameters are randomly sampled several times, and the test that provides the best split, i.e. the maximum amount of information gain, is chosen. This approach is particularly useful if the feature space is very large.

7.2.3 Randomized Decision Forest for Hand Pose Estimation

An RDF is an ensemble of RDTs trained on the same or slightly different datasets. The input to an RDF is a depth image I , and a pixel location \mathbf{x} . The output is a set of posterior probabilities for each hand part label c_i .

We use the same features as in [2]. Given a depth image $I(\mathbf{x})$, where \mathbf{x} denotes location, we define a feature $F_{u,v}(I, \mathbf{x})$ as follows:

$$F_{u,v}(I, \mathbf{x}) = I\left(\mathbf{x} + \frac{\mathbf{u}}{I(\mathbf{x})}\right) - I\left(\mathbf{x} + \frac{\mathbf{v}}{I(\mathbf{x})}\right). \quad (7.2)$$

The offsets \mathbf{u} and \mathbf{v} are relative to the pixel in question, and normalized according to the depth at \mathbf{x} . This ensures that the features are 3D translation invariant. Note that they are neither rotation nor scale invariant, and the training images should be generated accordingly. The depth of background pixels and the exterior of the image are taken to be a large constant.

Each split node is associated with the offsets \mathbf{u} and \mathbf{v} and a depth threshold τ . The data is split into two sets as follows:

$$C_L(\mathbf{u}, \mathbf{v}, \tau) = \{(I, \mathbf{x}) | F_{u,v}(I, \mathbf{x}) < \tau\} \quad (7.3)$$

$$C_R(\mathbf{u}, \mathbf{v}, \tau) = \{(I, \mathbf{x}) | F_{u,v}(I, \mathbf{x}) \geq \tau\}. \quad (7.4)$$

Here, C_L and C_R are the mutually exclusive sets of pixels assigned to the left and right children of the split node, respectively.

In the training phase, each split node randomly selects a set of features, partitions the data accordingly and chooses the feature that splits the data best. Each split is

scored by the total decrease in the entropy of the label distribution of the data:

$$S(\mathbf{u}, \mathbf{v}, \tau) = H(C) - \sum_{s \in \{L, R\}} \frac{|C_s(\mathbf{u}, \mathbf{v}, \tau)|}{|C|} H(C_s(\mathbf{u}, \mathbf{v}, \tau)) \quad (7.5)$$

where $H(K)$ is the Shannon entropy estimated using the normalized histogram of the labels in the sample set K . The process ends when the leaf nodes are reached. Each leaf node is then associated with the normalized histogram of the labels estimated from the pixels reaching it.

Starting at the root node of each RDF, each pixel (I, \mathbf{x}) is assigned either to the left or the right child until a leaf node is reached. There, each pixel is assigned a set of posterior probabilities $P(c_i|I, \mathbf{x})$ for each hand part class c_i . For the final decision, the posterior probabilities estimated by all the trees in the ensemble are averaged:

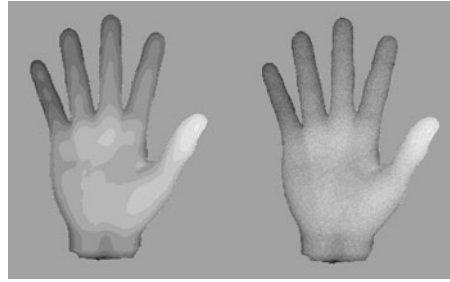
$$P(c_i|I, \mathbf{x}) = \frac{1}{N} \sum_{n=1}^N P_n(c_i|I, \mathbf{x}) \quad (7.6)$$

where N is the number of trees in the ensemble, and $P_n(c_i|I, \mathbf{x})$ is the posterior probability of the pixel estimated by the tree with index n . Another option is to multiply the posteriors. However, the trees are correlated, and multiplication is more prone to the effects of noise.

The RDF assigns each leaf node a set of posterior probabilities by counting the number of training pixels from every label that reach that node. This approach poses a balance problem if the number of training pixels significantly differ for labels. Indeed, the number of training pixels for the palm is several orders of magnitude larger than that of the finger tips. Hence, even a small portion of pixels from the palm area dominates the posterior likelihoods of the leaf nodes it reaches. One solution is to increase tree depth until all the leaves are pure. However, this causes overtraining or over-confident posteriors, and reduces performance on test set. To prevent this, (i) we stratify the sampling process and ensure that an almost equal number of pixels from each label are used for training; (ii) we stop the splitting process if fewer than a certain number of pixels are assigned to a node. This prevents overfitting and allows better generalization.

Since only synthetic images are used for training, the RDFs must also generalize to real data as retrieved by the depth sensor. To ensure this, and to prevent the RDF from memorizing artifacts or patterns associated with synthetic images, we perturb both the skeletal configurations and the generated depth maps. In particular, Gaussian noise is applied to each angle in the skeleton as well as to the depth information. The Gaussian noise is applied to each depth pixel separately. The effect of the resulting depth noise is depicted in Fig. 7.4. The image on the left is a synthetic depth image. Here, the contrast is maximized to make the artifacts visible. Note that the actual noise model of Kinect is very complex due to underlying algorithms. Here, Gaussian noise is not applied to imitate the Kinect sensor, but rather to prevent the RDF from memorizing the very precise depth information provided by a single

Fig. 7.4 The effect of added depth noise. The *image on the left* is the original synthetic depth image. The *image on the right* is the same image with added Gaussian noise ($\sigma = 3$)



digital hand model, and to better generalize to the slightly different hand shapes of actual people.

7.2.4 Joint Position Estimation

After each pixel is assigned posterior probabilities, the result is used to estimate the joint positions. To locate the actual joint coordinates, a number of approaches can be employed, such as calculating the centroid of all the pixels belonging to a hand part. However, finding the centroid is not robust against outliers, which is especially a greater problem for smaller hand parts.

To reduce the effect of outliers, the mean shift local mode finding algorithm [22] is preferred over finding the global centroid of the points belonging to the same class. The mean shift algorithm estimates the probability density of each class label with weighted Gaussian kernels placed on each sample. Each weight is set to be the pixel's posterior probability $P(c_i|I, \mathbf{x})$ corresponding to the class label c_i , times the square of the depth of the pixel, which is an estimate of the area the pixel covers, indicating its importance. The joint locations estimated using this method are on the surface of the hand and need to be pushed back to find an exact match for the actual skeleton.

Starting from a point estimate, or *seed*, the mean shift algorithm uses a gradient ascent approach to locate the nearest mode of the distribution. As the maxima are local, several different starting points are used and the one converging to the maximum score is selected. Finally, a decision regarding the visibility of the joint is made by thresholding the highest score reached during the mean shift phase. The joint positions estimated in this manner are then connected according to their configuration in the hand skeleton, forming the final pose estimate.

At this point, it is possible to make use of temporal or spatial information to infer a better skeleton estimate. For instance, a particle filter can be used to eliminate sudden jumps in joint locations, and skeletal constraints can be used to disregard some of the local maxima reached by the mean shift phase. An important constraint is that the joints on a finger lie on a 3D plane, which can also be used to detect occluded joints.

7.3 Experiments

Here, we report quantitative results for the hand pose estimation and hand shape classification methods. First, we introduce the synthetic and real datasets used in experiments.

7.3.1 Datasets

7.3.1.1 Synthetic Dataset

Performance of RDFs on previously unseen poses depends heavily on the training set provided. Ideally, we want the trained RDF to generalize to all possible hand poses. However, the number of images that need to be synthesized for this ambitious task is immense. A static hand pose is a single configuration of the 22-dof skeleton. The number of possible configurations, even with a modest step size for each angle, is huge. Moreover, simply rotating a single static pose in 3D to generate all possible views with a step size of 15 degrees, produces 15k images per pose. This suggests that the target application should determine the extent of the dataset. Here, we choose the 24 static ASL letters, the 10 ASL digits, and six hand poses that are widely known and used, such as the sign for OK. For the 40 poses selected and manually synthesized with the hand model, we rotate the camera in 3D, perturb the angles, and interpolate between the poses to generate 200k synthetic images. The offline learning method of Sect. 7.2 can be used to train an RDF on this dataset. However, to incorporate a larger dataset, incremental learning methods should be preferred [23].

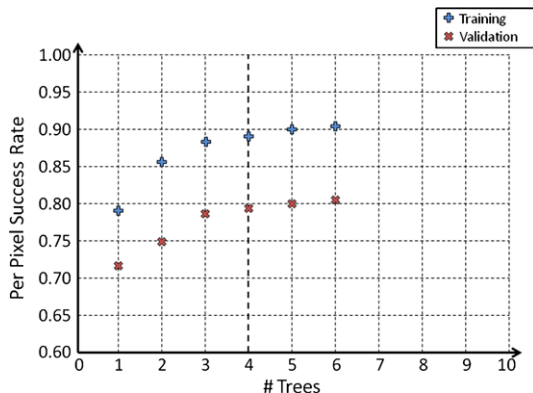
7.3.1.2 Real Dataset

For the hand shape classification task, both synthetic and real images can be used. However, only the accuracy on a real set is of importance. Therefore, a dataset consisting of real depth images retrieved from a Kinect depth camera is collected. Data collection is simple; one needs to perform the sign for several seconds in front of the sensor, while slightly moving and rotating the hand. We collected a dataset for the ten ASL digits from five different people. Each shot takes ten seconds, amounting to a total of 300 frames for each digit per person. Hence, the dataset consists of 15k images.

7.3.2 Effect of Model Parameters

The RDF parameters that have an effect on the classification accuracy are as follows: (i) The number of trees; (ii) the tree depth; (iii) the limits of \mathbf{u} , \mathbf{v} and τ ; (iv) the number of feature samples tried at each node; (v) mean shift weight threshold; (vi) the number of mean shift seeds.

Fig. 7.5 The effect of the forest size on the test accuracy



7.3.2.1 The Effect of the Forest Size

Training a large RDT by maximizing information gain is likely to produce overconfident posteriors. Since posterior probabilities are averaged over all trees, increasing the forest size produces smoother posteriors, alleviates overtraining, and allows better generalization, while monotonously increasing test accuracy. This is illustrated in Fig. 7.5. The trade-off is the linear increase in memory and the time it takes to test. We typically use one to four trees, as real-time performance is of importance in most application areas.

7.3.2.2 The Effect of the Tree Depth

The depth of a tree determines the number of tests to apply to the input. If the depth is too large, noisy training data will be isolated by the tests, causing overtraining. Likewise, a shallow tree will produce low-confidence, high entropy posteriors. Therefore, it is important to optimize the tree depth.

The effect of the tree depth is illustrated in Fig. 7.6. Overtraining starts at around depth 22, and the gain from increasing depth over 20 is minimal. As the need for memory increases exponentially, we prefer setting the depth to 20.

In our implementation, a tree of depth D evaluates pixels using exactly D binary comparisons. The number of internal nodes is $2^D - 1$, and the number of leaves is 2^D .

7.3.2.3 The Effect of the Feature Space

The feature space is determined by the maximum range of the offset parameters u , v and τ . We use a single limit for both x and y coordinates of the u and the v parameters, and a separate limit for the τ parameter. This defines the spatial context that can be used for tests in the form of a cube around the pixel. Intuitively,

Fig. 7.6 The effect of the tree depth on the test accuracy

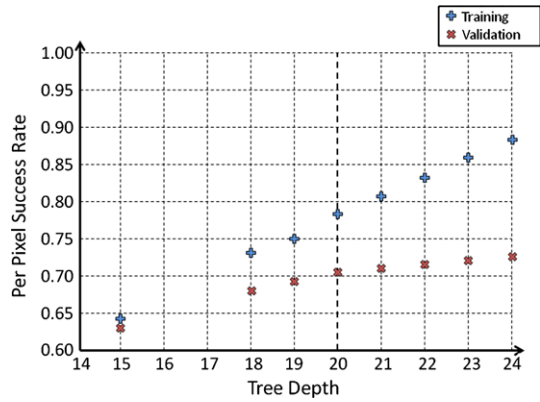
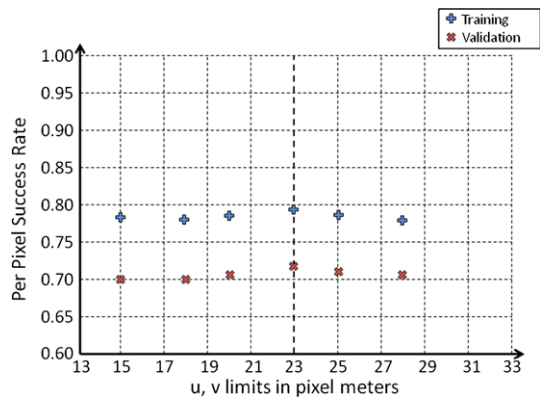


Fig. 7.7 The effect of the limits of the offset parameters u and v on the test accuracy

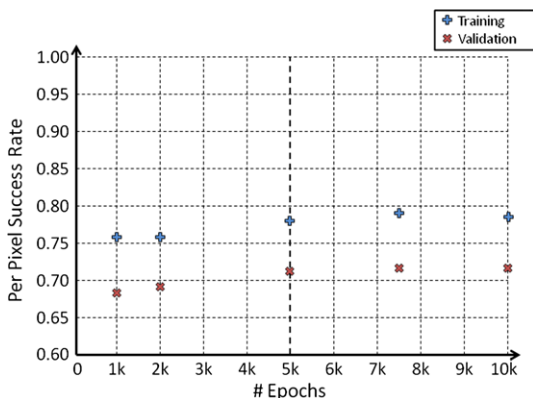


taking a larger context into account should increase the test accuracy. However, a fixed number of parameter values are sampled at each node. Hence, incorporating a larger context may reduce the probability of selecting good features that maximize information gain for a split. Moreover, the training dataset must be large enough to prevent the RDT from overtraining, if it uses a large spatial context. This effect is visible for different values of u and v limits in Fig. 7.7. The optimum value for the limit of u and v is estimated to be 23 pixel meters, i.e. 23 pixels if the hand is 1 m away, 46 pixels if the hand is 50 cm away, or 11.5 pixels if the hand is 2 m away from the camera. In our tests, we estimated the optimum value of τ to be 60 mm.

7.3.2.4 The Effect of the Sample Size

The sample size is the number of parameter values sampled from the feature space for each internal node. Increasing the sample size increases the test accuracy, as it is more likely to sample features that increase the information gain with a larger sample size. The trade-off is the increase in training time. Since forest size must

Fig. 7.8 The effect of the sample size on the test accuracy



be small due to memory constraints, the RDTs must produce confident posteriors. However, as we are sampling from a fixed feature space, the effect of the sample size levels off after some value. This is illustrated in Fig. 7.8. For the fixed limit of 23 pixel meters for \mathbf{u} and \mathbf{v} , the gain from increasing the number of trials is negligible after sample size reaches 5000.

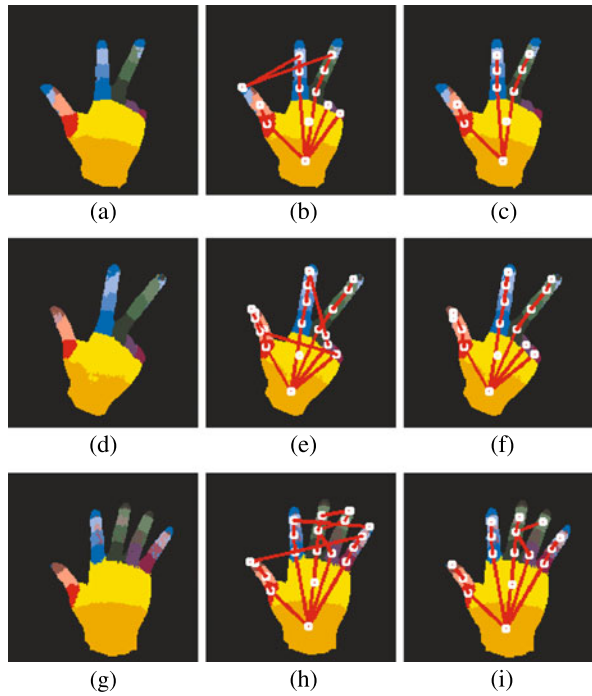
7.3.2.5 The Effect of the Mean Shift Parameters

Since hands are highly articulate and flexible objects, self-occlusion of entire hand parts is natural and happens frequently. In the ideal case, there should be no pixel assigned to the hidden hand part. However, it is common that some pixels are misclassified. In such cases, the mean shift algorithm determines the joint location for the hidden hand part based on the misclassified pixels only. Therefore, such spurious joint estimates need to be eliminated.

A decision regarding the visibility of the joint is made by thresholding the highest score reached during the mean shift phase. The effect of the thresholding process is shown in Fig. 7.9. Here, the images on the first column are the original pixel classification results, with colors assigned according to the highest posterior. The images in the second column are the same images, with the corresponding joint locations as estimated by the mean shift algorithm. The images on the third column are produced by eliminating joints that have low confidence values. Here, the confidence is defined as the value of the peak reached during the mean shift phase, which is evaluated using a combination of the pixel posteriors, joint bandwidth, which is a measure of the spread of the joint, and the importance of the pixel, which is the square of its depth, i.e. a measure of its area. The range of values depends on the implementation, and we empirically estimated it to be around 0.4. In the upper row of Fig. 7.9, the threshold is set to 0.5, which eliminates legitimate joints. In the middle row, the threshold is set to 0.4, and only spurious joints are eliminated. In the lower row, the threshold is set to 0.2, leaving one spurious joint intact.

Mean shift is a local mode finding method that only finds the closest maximum. To increase the likelihood of converging to the global maximum, we start multiple

Fig. 7.9 In the *upper row*, the confidence score threshold is set too high (0.5), eliminating true joints. In the *middle row*, the threshold is set correctly (0.4) and only the spurious joints are eliminated. In the *lower row*, the threshold is set too low (0.2), leaving one spurious joint intact



times from different seed points. The maximum with the highest score is selected, once all iterations converge. Seeds are randomly selected from the list of pixels with posterior probabilities higher than a certain value. We empirically determined this likelihood to be 0.35. The effect of the number of seeds is illustrated in Fig. 7.10. Here, the rows depict two examples, and the columns correspond to seed numbers 1, 2, 3 and 4, respectively. The higher this number, the more likely it is to converge to the correct joint locations. The trade-off is the increase in joint estimation time. In practice, we start from up to 20 different seeds.

7.3.3 Hand Pose Estimation Results

A synthetic dataset of size 200k formed with 40 hand poses is used to conduct hand pose estimation experiments. First, 5×2 cross-validation strategy is used to determine the best parameters. In this method, the dataset is randomly divided into two sets. In the first run, the model is trained on the first set and tested on the second set. In the second run, the model is trained on the second set and tested on the first set. This procedure is repeated on five randomly created pairs, and the average accuracy is regarded as a robust estimation of the success rate. The optimal forests are achieved with the parameters reported in Sect. 7.2: (i) Forest size = 4; (ii) Tree depth = 20; (iii) Offset limit = 23 pixel-meters; (iv) Sample size = 5000;

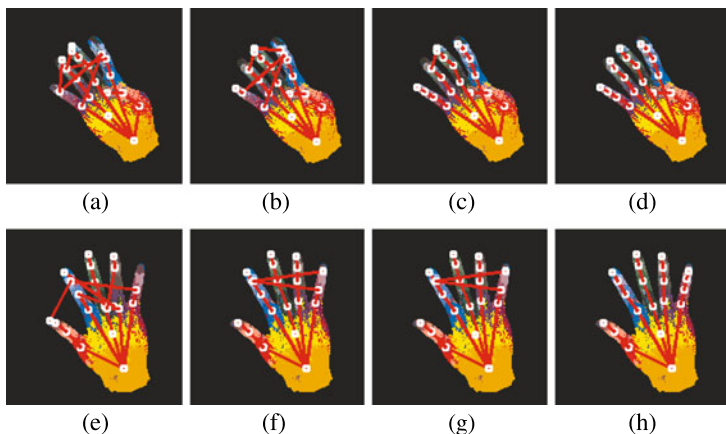


Fig. 7.10 The effect of the number of starting points for the mean shift algorithm. The *columns* correspond to number of seeds 1, 2, 3 and 4, respectively

(v) Mean shift seed posterior threshold = 0.35; (vi) Number of seeds = 20. The test accuracy of the resulting RDF is also determined with 5×2 cross-validation. The per-pixel classification accuracy (using hard labels) on this dataset is 67.5 %.

Another important measure of error is the average distance between the estimated joint coordinates and the ground truth. However, spurious joints, especially misplaced finger tips, have a large effect on this type of error. Therefore, we estimate the number of spurious or missing joints as an indicator of the error instead. Hence, we count the number of correct joints in the test dataset, and divide it over the number of visible joints. The visibility of the joints is determined automatically, and correctness of a joint estimation is determined by thresholding the projected distance between the estimated and actual joint coordinates. For the synthetic dataset of size 200k, with 40 poses, 82.1 % of the visible joints are estimated correctly. For a smaller dataset of size 20k, formed by ASL digits only, the method is able to find 97.3 % of the joints correctly. Most of the error in the latter case is attributable to misplaced finger tips.

7.3.4 Proof of Concept: American Sign Language Digit Recognizer

To test the system on a real world application, we developed a framework for classifying ASL digits in real time. The method described in Sect. 7.2 gives estimates of the hand skeleton as output. The pose classifier uses these estimates to recognize the digits by mapping the estimated joint coordinates to known hand poses.

First, the RDF is trained on a synthetic ASL digit dataset of size 20k, so that it learns to extract the skeleton from poses that closely resemble ASL digits. Then, this RDF is used to evaluate the real depth images acquired from the Kinect, while a

Table 7.1 Classification rates and evaluation times of each classifier on the ASL digit dataset consisting of 20k synthetic images

| Method name | Accuracy | Classification duration (ms) |
|-------------|----------|------------------------------|
| ANN | 99.89 | 0.0045 |
| SVM | 99.96 | 0.3 |

user is performing ASL digits. A training set is formed using the extracted skeleton parameters by properly labeling each hand skeleton according to its corresponding hand shape. Such a training set can be used to train classifiers in a supervised manner. These shape classifiers can then be used to map the extracted hand skeletons into ASL digits in real time.

7.3.4.1 Hand Shape Classifiers

As the intended usage of the system is real-time recognition of ASL digits, speed is as important as the recognition rate. We choose artificial neural networks (ANN), since they are fast, and SVMs, since they are accurate. We use 5×2 cross-validation strategy for both model selection and to test accuracy. Model selection for the RDF is done only on the synthetic dataset.

7.3.4.2 Model Selection on the Synthetic Dataset

Both the RDFs and the skeleton classifiers need to be optimized. To select an RDF model, a synthetic dataset needs to be used, since there is no ground truth labels that are associated with real data. We optimized ANN and SVM separately for both synthetic and real datasets. The synthetic dataset consists of 20k samples, formed by 2k synthetic images for each of the ASL digits. The RDF parameters are as reported in Sect. 7.3.3. For the ANN, the optimum number of hidden nodes is estimated to be 20. For SVMs, the optimal parameters are found to be 2^6 for the cost parameter and 2^{-4} for the Gaussian spread (γ).

The test accuracies and evaluation times are listed in Table 7.1. The first column gives the average accuracies achieved by the cross-validation tests. The second column gives the evaluation times. Evidently, ANN is significantly faster than SVM. However, SVM performs slightly better on the test data. The intermediate phases and the final skeletons for several examples are given in Fig. 7.11.

7.3.4.3 ASL Digit Classification Results on Real Data

We conducted 5×2 cross-validation and grid search to estimate the optimal parameters of ANN and SVM again for the real dataset. Table 7.2 shows the parameters tested.

Fig. 7.11 Examples of extracted skeletons on synthetic ASL images. *Upper row* lists the depth images. *Middle row* shows the per pixel classification results. *Third row* displays the estimated joint locations on top of the labeled images. The finalized skeleton is shown in the *bottom row*

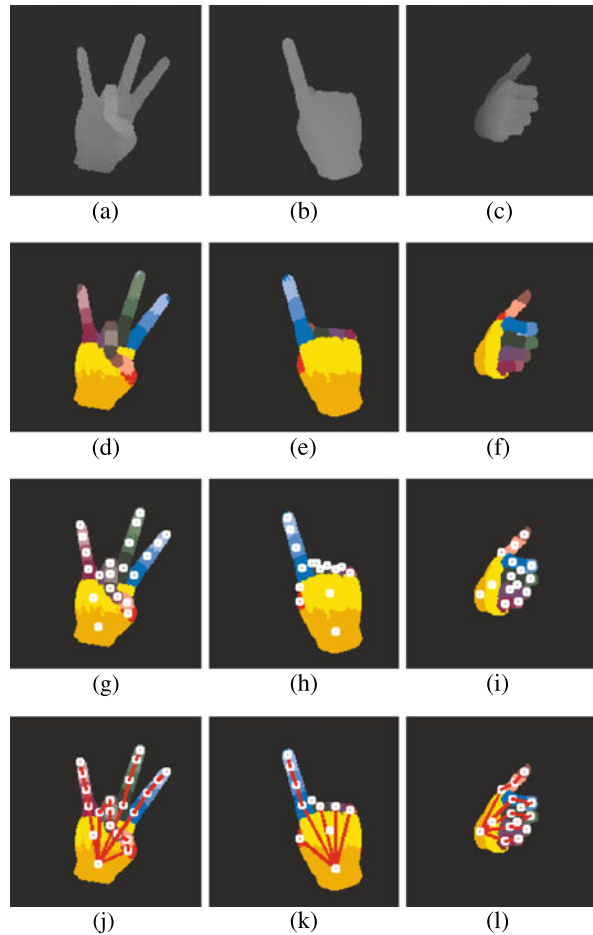


Table 7.3 lists the optimal parameters and recognition rates on training and validation sets for ANNs and SVMs for real data. SVMs outperform ANNs and reach nearly perfect accuracy on the validation set, indicating that the descriptive power of the estimated skeleton is sufficient for the task of hand shape classification on real depth images.

7.4 Conclusion

In this study, we described a depth image-based real-time skeleton fitting algorithm for the hand, using RDFs to classify depth image pixels into hand parts. To produce the huge amount of samples that are needed to train the decision trees, we developed a tool to generate realistic synthetic hand images. Our experiments showed that the system can generalize well when trained on synthetic data, backing up the

Table 7.2 Tested parameter values (H : hidden nodes, C : SVM cost, γ : Gaussian spread)

| Method name | Parameter values |
|-------------|---|
| ANN | $H = \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55\}$ |
| SVM | $C = \{2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7\}$ |
| SVM | $\gamma = \{2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1\}$ |

Table 7.3 Optimal parameters, average training and validation accuracies

| Method name | Optimal parameters | Training accuracy | Validation accuracy |
|-------------|----------------------------------|-------------------|---------------------|
| ANN | Hidden nodes = 40 | 99.27 | 98.81 |
| SVM | Cost = 2^5 , $\Gamma = 2^{-2}$ | 100 | 99.90 |

claims of Shotton et al. in [2]. In particular, just by feeding manually designed hand poses corresponding to ASL digits to the RDFs, the system learned how to correctly classify the hand parts for real depth images of hand poses that are close enough. This in turn enabled us to collect real data labeled by the RDFs that can be used for further pose classification tasks. We demonstrated the efficiency of this approach by reaching a recognition rate of 99.9 % using SVMs on real depth images retrieved with Kinect. The features used by SVMs are the mean shift-based joint estimates calculated in real time from the per pixel classification results.

We focused on optimizing the speed and accuracy of the system, in particular by performing grid search over all model parameters. The resulting framework is capable of retrieving images from Kinect, applying per pixel classification using RDFs, estimating the joint locations from several hypotheses in the mean shift phase, and finally using these locations for pose classification in real time. The system is optimized for multicore systems and is capable of running on high end notebook PCs without experiencing frame drops. Further enhancement is possible through the utilization of the GPU, as described in [24], and this framework can be used along with more CPU intensive applications such as games and modeling tools. This method is one of the first to retrieve the full hand skeleton in real time using a standard PC and a depth sensor, and has the extra benefit of not being affected by illumination.

The main focus of this study is skeleton fitting to the hands from a single frame. Consequently, temporal information is ignored, which can certainly be used to enhance the quality of the fitted skeleton, via methods such as Kalman [25] or particle filtering [26].

References

1. Microsoft Corp. Redmond, WA. Kinect for Xbox 360
2. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: IEEE Conference on Computer Vision and Pattern Recognition (2011)

3. Girshick, R., Shotton, J., Kohli, P., Criminisi, A., Fitzgibbon, A.: Efficient regression of general-activity human poses from depth images. In: *International Conference on Computer Vision* (2011)
4. Erol, A., Bebis, G., Nicolescu, M., Boyle, R.D., Twombly, X.: Vision-based hand pose estimation: a review. *Comput. Vis. Image Underst.* **108**, 52–73 (2007)
5. Athitsos, V., Sclaroff, S.: Estimating 3D hand pose from a cluttered image. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2003)
6. Romero, J., Kjellstrom, H., Kragic, D.: Monocular real-time 3D articulated hand pose estimation. In: *Humanoids*, pp. 87–92 (2009)
7. De Campos, T.E., Murray, D.W.: Regression-based hand pose estimation from multiple cameras. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2006)
8. Tipping, M.E., Smola, A.: Sparse Bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.* **1**, 211–244 (2001)
9. Rosales, R., Athitsos, V., Sigal, L., Sclaroff, S.: 3D hand pose reconstruction using specialized mappings. In: *International Conference on Computer Vision* (2001)
10. Stergiopoulou, E., Papamarkos, N.: Hand gesture recognition using a neural network shape fitting technique. *Eng. Appl. Artif. Intell.* **22**, 1141–1158 (2009)
11. De La Gorce, M., Fleet, D.J., Paragios, N.: Model-based 3D hand pose estimation from monocular video. *IEEE Trans. Pattern Anal. and Mach. Intell.*, Feb. 1–14 (2011)
12. Oikonomidis, I., Kyriazis, N., Argyros, A.: Efficient model-based 3D tracking of hand articulations using Kinect. In: *British Machine Vision Conference* (2011)
13. Stenger, B., Mendonça, P.R.S., Cipolla, R.: Model-based 3D tracking of an articulated hand. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2001)
14. Bray, M., Koller-Meier, E., Van Gool, L.J.: Smart particle filtering for high-dimensional tracking. *Comput. Vis. Image Underst.* **106**, 116–129 (2007)
15. Heap, T., Hogg, D.: Towards 3D hand tracking using a deformable model. In: *International Conference on Automatic Face and Gesture Recognition*, pp. 140–145 (1996)
16. Mo, Z., Neumann, U.: Real-time hand pose recognition using low-resolution depth images. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2006)
17. Malassiotis, S., Srinivas, M.: Real-time hand posture recognition using range data. *Image Vis. Comput.* **26**, 1027–1037 (2008)
18. Liu, X., Fujimura, K.: Hand gesture recognition using depth data. In: *Automatic Face and Gesture Recognition* (2004)
19. Suryanarayan, P., Subramanian, A., Mandalapu, D.: Dynamic hand pose recognition using depth data. In: *International Conference on Pattern Recognition* (2010)
20. Breiman, L.: Random forests. *Mach. Learn.* **45**, 5–32 (2001)
21. Uebbersax, D., Gall, J., Van den Bergh, M., Van Gool, L.: Real-time sign language letter and word recognition from depth data. In: *International Conference on Computer Vision—Workshop on Human Computer Interaction: Real-Time Vision Aspects of Natural User Interfaces* (2011)
22. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. *Pattern Anal. Mach. Intell.* **24**, 603–619 (2002)
23. Basak, J.: Online adaptive decision trees: pattern classification and function approximation. *Neural Comput.* **18**, 2062–2101 (2006)
24. Sharp, T.: Implementing decision trees and forests on a GPU. In: *European Conference on Computer Vision* (2008)
25. Welch, G., Bishop, G.: *An Introduction to the Kalman Filter* (1995)
26. Isard, M., Blake, A.: CONDENSATION—conditional density propagation for visual tracking. *Int. J. Comput. Vis.* **29**, 5–28 (1998)